DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GOR/MA/88D-1

INFLUENCE DIAGRAMS: AUTOMATED ANALYSIS WITH DYNAMIC PROGRAMMING

THESIS

Christopher Thomas Baron
Captain, USAF

AFIT/GOR/MA/88D-1

Approved for public release; distribution unlimited

AFIT/GOR/MA/88D-1

INFLUENCE DIAGRAMS: AUTOMATED ANALYSIS WITH DYNAMIC PROGRAMMING

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Operations Research

Christopher Thomas Baron, B.S.
Captain, USAF

December 1988

Preface

I would like to thank my thesis advisor Captain Joseph Tatman for his enthusiasm and confidence in me and in the software package called AFids. Without his help and encouragement the software would certainly have been less in all ways. I would also like to thank my thesis reader Major Bruce Morlan for his many helpful suggestions.

To my wife Lori I can only say 'It's over.' and thank you for making the whole AFIT program much easier than it could have been. Your love and encouragement kept me going in the tough times. To my boys Michael, Patrick, and Danny, thank you for putting up with the use of 'your' computer for boring stuff like thesis writing, and the absence of your father during those many long nights I spent hunched over a computer screen or doing homework.

I would also like to thank Borland International for Turbo Pascal. Without this wonderful programming environment I doubt if AFids would have half the capability it has.

My sincere hope for the AFids software is that it will be used. For students and faculty to use the fruits of my labor for actual problem solving would provide for me much satisfaction.

Christopher Thomas Baron

Table of Contents

List of Figures

Abstract

The major goals of this thesis research were to develop a user friendly software package for processing influence diagrams, and to implement in software the extensions necessary for dynamic programming without special action or knowledge on the part of the user. The final goal was to demonstrate the efficiency of the dynamic programming techniques by applying them to several example problems.

A software package, AFids (AFIT influence diagram system) was developed. The system is capable of performance equivalent to the current state of the art in commercial influence diagram software. AFids incorporates the basic influence diagram operations, the separable value function extensions, and an algorithm to automatically solve any properly formed influence diagram. Separation of the value function is automatic and requires no action or special knowledge on the part of the user beyond representing the value function explicitly as a sum or product. The software uses menus, data entry screens, and graphics to provide an effective and friendly user interface. Several extensions to influence diagram theory were implemented in the AFids package including the concepts of value rounding and outcome limiting to control the combinatorial explosion encountered when processing chains of deterministic nodes. The system is cost free and available in either source or compiled form for government users. There are no restrictions on use or distribution except for commercial use. The software runs on MS-DOS compatible microcomputers and was programmed in Turbo Pascal. A users manual, and a description of the AFids data structures are provided for future users and researchers.

Two application examples are presented, demonstrating both the efficiency of the dynamic programming features and the limitations of influence diagrams in modeling problems with significant functional relations.

AFids provides a solid capability for processing influence diagrams throughout

the decision analysis cycle from formulation to solution. Inclusion of the separable value function and deterministic processing node functions represent advancements in influence diagram software and allow previously computationally intractable problems to be solved via influence diagrams. Finally, AFids will become the standard influence diagram software for the decision analysis curriculum at AFIT.

INFLUENCE DIAGRAMS: AUTOMATED ANALYSIS WITH DYNAMIC PROGRAMMING

I. Introduction

1.1 General Background

Influence diagrams are a recently developed graphical modeling tool for representing both the conceptual elements and the mathematical structure of a problem. An influence diagram may be used as a formal problem description suitable for manual or computer based solution and/or as an informal conceptual aid useful in helping a decision maker visualize the elements of the problem and their relationships and dependencies.

Influence diagrams were originally developed for decision analysis applications, helping decision makers choose the best alternative in problems with significant uncertainty or subjective factors. Originally conceived as an alternative to decision trees, other areas of application such as stochastic control, classical statistics, artificial intelligence, and other areas within operations research have emerged [1:1]. The dual nature of the problem representation using influence diagrams has some significant advantages.

- Problem variables are concisely and unambiguously represented in the graph as are the interrelationships between them.
- The full mathematical description of the problem is contained within the data portion of the diagram.
- The diagram may be easily stored and manipulated by computer allowing rapid solution and convenient 'what-if' query capability.

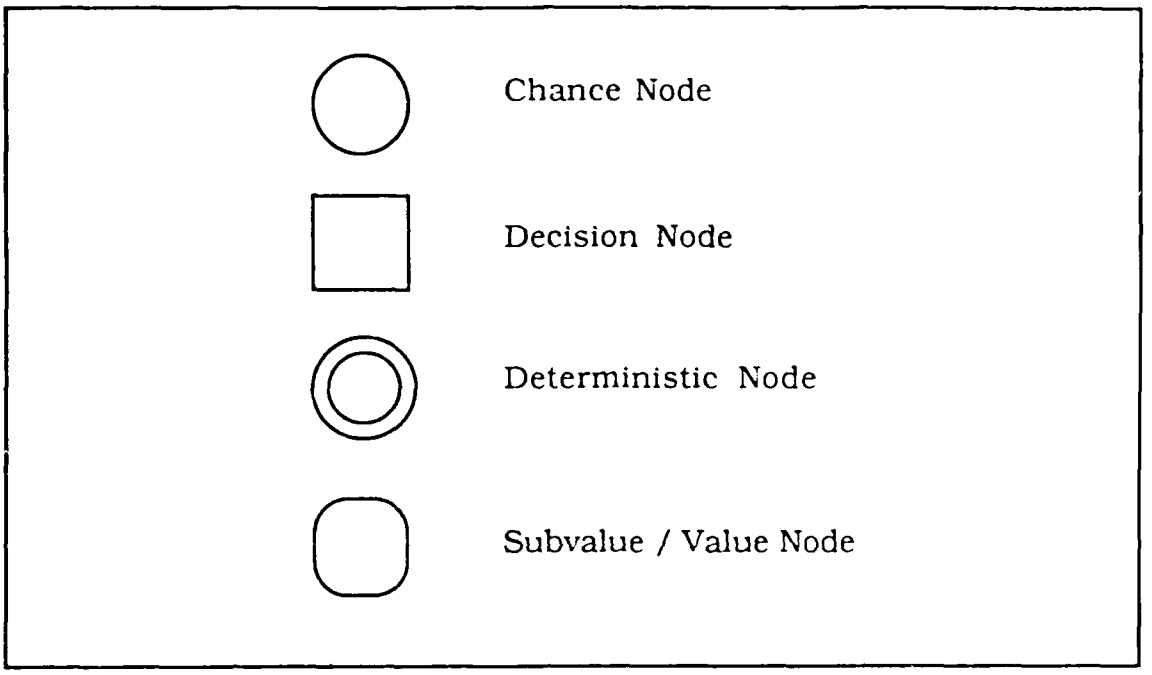

Figure 1. Types of Nodes

The format of an influence diagram is an acyclic graph, with nodes corresponding to the variables in the problem, and arcs connecting the nodes to represent probabilistic dependence or information flow. The nodes, of which there are several kinds, each have an associated frame of data describing the nature, outcomes, and for the chance nodes, the probability distribution, of the variable they represent. Arcs between nodes represent the relationships between the nodes and hence the variables they represent in the model [9:10]. The types of node included in basic influence diagrams are (see Figure 1):

Chance Node The chance node represents a random variable or event with some probability distribution describing the likelihood of its various outcomes. An arc between two chance nodes indicates probabilistic dependence. Currently available software implementations only allow discreet probability distributions for chance nodes.

Decision Node The decision node represents a choice among several alternatives.

One of the alternatives to be chosen will maximize the expected value of the model. An arc into a decision node indicates the outcome of the node at the head of the arc is known at the time the decision is made.

Deterministic Node The variable represented by a deterministic node has its value determined by a function of the values of the nodes preceding it in the diagram.

Value Node The value node is a special case of the deterministic node and represents the objective function of the model. Influence diagram models are generally formulated so that maximizing the value node outcome maximizes the variable of interest in the model. A typical value function might represent profit, or number of aircraft lost (the modeler could maximize the negative of aircraft lost to get a minimum number).

All of the chance node probability distributions taken together represent the joint probability distribution of the random variables in the problem.

Nodes may be eliminated from the diagram by performing certain value preserving transformations. The transformations such as, decision maximization, conditional expectation, and algebraic functional combination allow one or more nodes to be removed from the diagram while preserving the expected value and optimal decision policies of the model. This process of removing nodes reveals the optimum decision policy leading to the maximum expected value of the problem while taking into account the probabilities of the outcomes of the chance nodes [6:879]. Figure 2 shows an example diagram representing the classic text-book problem modeling an oil wildcatter's decision of whether or not to drill for oil. The implications of the arcs between the different types of nodes will be discussed further in Chapter 2.

A major shortcoming of the traditional influence diagram is that the separable nature, if any, of the value function cannot be exploited [7:3]. A large number of the decisions that must be made both in theoretical and practical situations resemble

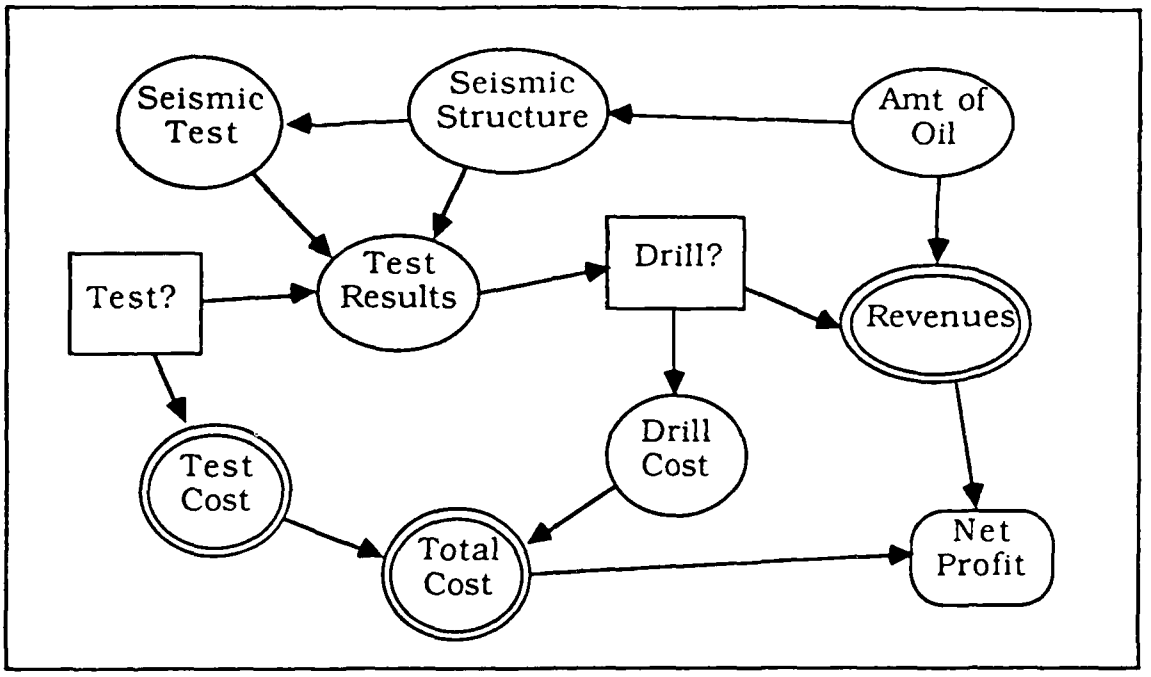

Figure 2. An Example Influence Diagram

Markov decision processes [8:11]. The characteristic properties of a Markov decision process are: the separability of the objective function into stages, and the Markovian assumption that each stage may be optimized independently of other stages preceding the current stage in time. The justification for using dynamic programming in influence diagrams lies in the enormous computational efficiency that solution via dynamic programming can provide. For example, in a problem of finding the shortest path between two points with 20 intermediate points, the brute force method of explicit enumeration of all possible routes would require more than 3×10^6 additions and 184,000 comparisons to find the optimum answer, while dynamic programming can solve the problem with only 220 additions and 100 comparisons [2:9]. Obviously a whole new class of problems which were formerly computationally intractable for influence diagrams become solvable when a dynamic programming capability is implemented.

Dynamic Programming is an optimization procedure that is computationally efficient when applied to problems having a sequence of related decisions or stages

[2:1]. Some examples include yearly budget allocations, program scheduling, and minimum path planning. The goal of dynamic programming is then, to select the sequence of decisions which yield the maximum expected value for the problem.

1.2 Specific Research Goals

The three major goals of this thesis research were:

1. To develop user friendly software for processing influence diagrams employing a graphical user interface and menu driven command structure.

   The current influence diagram software used at AFIT, PerForma [1], was written in an interpreted dialect of the LISP programming language. It provides only a rudimentary user interface, consisting of a set of LISP functions called from the interpreter command line, requiring the user to enter confusing LISP syntax commands. No graphical representation of the diagram is provided and diagram processing is extremely slow due to the interpreted nature of the language. No method for automatically solving influence diagrams was included in PerForma, requiring the user to manually determine the steps needed to process the diagram.

   The goal then, was to develop a software package employing a graphical interface to represent the diagram visually while providing fast processing of the diagram with a user friendly command structure. Ideally the software should be written in a compiled, high level procedural language to combine processing speed with ease of modification. The software should implement the automatic diagram solving algorithm allowing problems to be solved without requiring the user to concentrate on the individual steps needed to reach the solution. Finally, the software should be of such quality that it could be delivered to government customers sponsoring research along with a model of their problem suitable for use at the customers location, i.e. it must be more than just a research tool.

2. The software, incorporating the basic influence diagram operations, should be extended to include the additional operations and node types defined in Tatman [8]. The primary concept in these extensions is to allow the separation of the value or objective function of the problem into parts or stages which may be solved separately. This separation can significantly reduce the dimensionality of a problem and is most commonly seen in dynamic programming. The software should ideally recognize a separable value function and perform these operations without special effort or knowledge on the part of the user. These concepts are discussed in detail in Chapter 2. A modification to the standard solution algorithm is also required.

3. It is desirable to demonstrate the efficiency of the software and its extensions by applying them to several example problems.

1.3 Review of Influence Diagram Literature

The concept of influence diagrams was originated by Merkhofer, Miller, Howard, Matheson and Rice in about 1976 [6:871]. One of the first attempts to automate the transition from influence diagram formulation to analysis was by Howard and Matheson in 1981 [3]. While only chance and decision nodes were included in Howard and Matheson's conceptualization, most of the other important features of today's influence diagram language were considered including: arcs representing probabilistic and informational influences, the prohibition on loops in the diagram, the importance of the 'no-forgetting' arcs which must be added to ensure that outcomes from previous decisions and chance outcomes are available to succeeding nodes, and the advantages of the influence diagram versus decision tree representations [3:735-739].

Owen extended the rather theoretical applications views of Howard & Matheson to provide guidelines on the practical use of influence diagrams in formulating real-world decision problems and capturing the decision makers knowledge of the problem and then using that knowledge as an intimate part of the model construc-

tion process [5:766-777]. Owen also realized that the very form of the influence diagram helped to produce correctly formed and balanced models [5:768]. An area mentioned by Owen which has not received further attention in the literature is that of a mathematical characterization of the strength of the influence between variables.

One of the truly seminal works in the influence diagram field was written by Shachter in 1984 [6]. This one short article presented the rigorous mathematical basis for all of the node types and diagram operations as well as an algorithm guaranteed to solve a properly formed diagram [6:879]. It is interesting to contrast the change in tone given in the conclusions of the Howard and Matheson paper of 1981 with that of Shachter in 1984. Howard and Matheson are presenting a new idea about how to represent decision problems: "We have shown how influence diagrams can be used to model the primary decision problem ..." [3:762]. Shachter, on the other hand, only three years later, is formally defining the operations to be used in real world applications: "The influence diagram has become a useful tool for analysts in communicating with decision makers and experts. ..." [6:882].

The Phd dissertation of Tatman [8], one of Shachter's students, is the final step in influence diagram modeling to date (at least in the direction of the author's thesis research) [8]. This work lays the foundation for the inclusion of dynamic programming in influence diagrams. [8:148]. The important areas recognized are: the limitations of restricting separable value functions to a single node, the need for an additional node type (the subvalue node) to efficiently handle dynamic programming processes, the requirement that the solving system recognize a separable value function and be able to represent that function utilizing the new node type, and that a modification to the solving algorithm was needed [8:16]. The conclusion to the dissertation clearly leads to the author's thesis research with the following statement: "The influence diagram with subvalue nodes is an effective tool for formulating and analyzing decision problems with separable value functions. ..." [8:147]

1.4 Literature Review Conclusions

Influence diagrams have progressed rapidly from a mathematical novelty in 1981 to a well understood and widely recognized problem modeling/solution technique in 1988. The early work of Howard, Matheson, and Shachter provide a solid foundation upon which the theory and practical applications of influence diagrams are built. Several of Shachter's students have continued the influence diagram evolution by extending influence diagram solution techniques and developing aids and methods for utilizing the influence diagram throughout the entire problem solving process.

Dynamic programming has long been recognized to convey significant computational efficiency on certain types of problems and the work of Tatman brings the promise of that efficiency to influence diagrams.

1.5 Review of Influence Diagram Software

Currently there are two software packages for processing influence diagrams (besides the author's) available. They are DAVID, a commercial package distributed by Duke University and written by Ross Shachter, and PerForma which is owned by AFIT and was written by Thomas Burwell and Joseph Tatman.

DAVID is a state of the art influence diagram/decision analysis package for the Apple Computer *Macintosh* series of microcomputers. The software has the most extensive analysis features currently available, including [1:8]:

1. Graphical display of the diagram and a graphical user interface for manipulating both the appearance of the graph and the solving operations.
2. Sensitivity analysis for uncertainties, risk tolerance, and the value of information.
3. Primitive dynamic programming functions.
4. Value lotteries for policy comparison.

5. An option for a function to derive discreet probability distributions.

6. Graphical display of probability distributions and sensitivity plots.

DAVID does have some serious limitations however, first, DAVID requires at least one mega-byte of RAM memory to execute (and seems to need 2 mega-bytes to process other than trivial problems); this excludes the vast majority of Macintosh computers from its use, second, program execution is slow due to the program being written in a compiled version of LISP, third, the availability of Macintosh computers in most government agencies is severely limited, and fourth, no source code is provided for possible modification nor is DAVID freely distributable to customers.

PerForma was written by a former AFIT student Thomas Burwell [1]. The program is written in interpreted LISP code and as might be expected executes very slowly. PerForma includes the basic node removal and diagram manipulation operations implemented as a set of LISP functions. Since AFIT owns the source code PerForma may be freely distributed to customers and modified as needed. However, no user interface is provided and no graphical display of the diagram is available. Shachter's solving algorithm is not implemented nor are any of the sensitivity analysis and value modeling features of DAVID. PerForma is basically a research tool suitable for small problems and class homework assignments.

There is then, a definite need for an influence diagram software package to fill the gap between DAVID, which has many features but is expensive and commercially restricted, and PerForma, which has primitive features, is cost free and is unrestricted in use and modification. The software developed in conjunction with this thesis research fills this gap. It is discussed in detail in Chapter 3.

II. Influence Diagrams and Dynamic Programming

This section will discuss the significance of the graph and data structures to the influence diagram, and the fundamental influence diagram reduction operations. As previously stated, an influence diagram is a two level hierarchical representation of a problem, its component factors or variables, and their relationships.

2.1 Basic Influence Diagram Concepts and Operations

The top level (the graph) visually represents the problem such that each object (node or arc) in the graph maps to an object or relationship in the second or data level. Thus, an influence diagram is mathematically precise in its representation and may be directly manipulated via value preserving reduction operations. Each operation corresponds directly with an operation in classical probability calculus. These operations will reveal the optimum decision policy for decision nodes and the expected value of the value node. Figure 3 shows the basic node types along with the unique data structures that each contains. An algorithm has been developed and theoretically proven able to reduce any well formed influence diagram down to a single node (the value node) which has as its value the expected value of the model [6:879].

Arcs in the diagram also have specific meanings. Arcs leaving a node (the predecessor node) and entering a chance node (the successor node) are called conditioning arcs and *may* represent conditional dependence of the successor node's probability distribution on the outcomes of the predecessor node. Conditional dependence is not guaranteed by the presence of an arc into a chance node. However, arcs into chance nodes where no dependence exists serve no useful purpose in the diagram, they increase the computational size of the solution, and can lead to false perceptions of dependence on the part of decision makers and decision analysts and thus are to be avoided. Lack of an arc into a chance node definitely indicates prob-

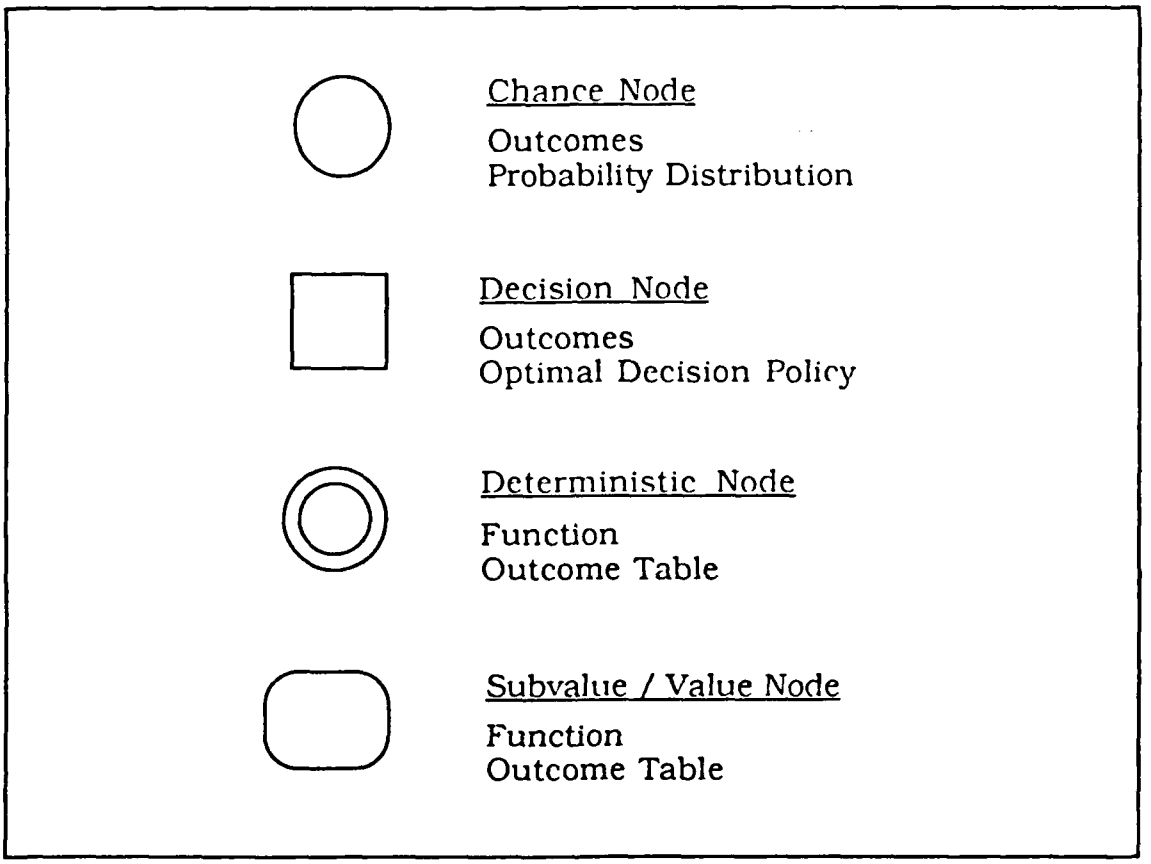

Figure 3. Influence Diagram Node Types and Their Data Elements

abilistic independence between the two nodes. For example, if there is an arc from some node y to a chance node x then the probability of a given outcome of x occurring probably depends on the outcome of node y. If however, there is not an arc between x and y then the outcome probabilities of x **do not** depend on the outcome of y.

An arc entering a decision node indicates information known at the time the decision represented by the node is made. For example, if a stock broker is to make a decision whether to buy or sell stock on a given day he knows the closing price for the stock on the previous day. In an influence diagram this information flow would be depicted as an arc from chance node *yesterday–close* to decision node *buy–or–sell?*. A logical assumption is that the decision maker does not forget any decision made previously nor does he forget any information available at the time the previous decision was made. This 'no forgetting' is represented by arcs from the previous decision nodes to successor decision nodes either direct or indirect. 'No forgetting' arcs are also added between a previous decision nodes direct predecessors and successor decision nodes. Often these arcs are left out when drawing the diagram but they must be inserted either explicitly or by the computer before solving can begin.

Since a deterministic node's outcome values are determined by the outcome values of its predecessors, an arc into a deterministic node indicates information dependence. The outcomes of all a deterministic node's predecessors must be defined in order for the deterministic node's outcomes to be calculated. Arcs into a value node have the same meaning as for a deterministic node since the former is simply a special case of the later.

In order to determine the expected value of an influence diagram model, all nodes except the value node must be removed. The value node will then contain the expected value for the model and the process of removing the decision nodes will reveal the decision policy which will maximize the problem expected value. There

are four basic node removal operations required to solve an influence diagram. They are:

Barren Node Removal A barren node is defined as any node (except the value node) which has no successors. A barren node may simply be removed from the diagram without affecting the problem outcome because the fact that it has no successors implies that it has no influence either directly or indirectly on the value node. In order to solve a diagram any barren nodes created must be removed after each reduction operation.

Expectation If a chance node directly precedes the value node and nothing else in a properly formed diagram it may be removed by conditional expectation. Expectation removes a node by summing the product of probabilities for the chance node's outcomes with the value node's value resulting from each outcome. A side effect is that all direct predecessors of the removed node are now direct predecessors of the value node.

Maximization If a decision node is a direct predecessor of the value node and all other direct predecessors of the value node are also informational predecessors of the decision node then the decision node may be removed by maximizing the expected value of the value function conditioned on the other predecessors of the value node. A side effect of maximization is that some of the informational predecessors of the decision node may become barren nodes since the value node does not inherit any new predecessors from this reduction.

Arc Reversal If an arc exists between two chance nodes and there is no other path between them then the arc may be reversed by applying Bayes rule to the two node's probability distributions. A side effect is that the two nodes involved inherit each others predecessors, possibly creating new arcs in the diagram. This operation is often needed when solving influence diagrams to allow a chance node to be removed by expectation. Bayes rule is also useful

in gaining insight into conditional probabilities. For example, a woman thinks she has a 50/50 chance of being pregnant. A pregnancy test is known to give a positive result with probability 0.99 if the woman is pregnant(Pr$test = pos$—$preg = true$), and to give a negative result with probability 0.90 if she is not (Pr$test = neg$—$preg = false$). The woman however, would like to know the probability that she is pregnant given the test result is positive (Pr$preg = true$—$test = pos$). This probability can be calculated by performing Bayes rule on the probability distribution of the test results and of her being pregnant as $\simeq 0.908$.

2.2 Basic Influence Diagram Solution Algorithm

This section describes briefly the general influence diagram solution algorithm as defined by Shachter [6:879]. As Shachter states:

> This section combines the basic transformations developed so far into a procedure that can evaluate any oriented [has a value node], regular [contains no cycles] influence diagram. The procedure will remove nodes from the diagram until only the value node remains. At that point, it has determined all of the optimal policies and computed the maximal expected utility [value].

The algorithm as given by Shachter is as follows:

DEFINE PROCEDURE ID-EVAL AS
BEGIN
  check for oriented, regular diagram
  add "no forgetting" arcs
  eliminate all barren nodes
  WHILE predecessors of the value node $\neq$ none DO
  BEGIN
    IF a chance node may be removed by expectation
    THEN remove the chance node

```
        ELSE IF a decision node may be removed by maximization
        THEN BEGIN
           remove the decision node
           remove any created barren nodes
        END
        ELSE BEGIN
           find a chance node which has the value node as a successor and
           only other chance nodes as its remaining successors
           WHILE chance successors exist DO
           BEGIN
              find a successor with no other path to the node
              reverse the arc between them
           END
           remove chance the chance node by expectation
        END
     END
  END
```

Since the algorithm will remove at least one node at every step the algorithm will always terminate.

2.3 Deterministic Node Processing

One important class of nodes is not mentioned in the above algorithm. Deterministic nodes were considered by Shachter to be only a special kind of chance node. This reasoning follows from the fact that a deterministic node may be transformed into a chance node by the following method.

1. Calculate all of the possible outcomes for the deterministic node. The number of outcomes will be:

$$\prod_{i=1}^{all\ preds} Outcomes(pred)_i$$

Note, this number grows exponentially with the number of predecessors.

2. Create a probability array such that, for a given combination of predecessors, the probability of the deterministic node outcome is equal to 1.0 for the deterministic function value the combination produced and 0.0 for all the others. The maximum size of the probability array will then be: Number of unique outcomes $\times$ the number of predecessor combinations.

3. Change the label on the node from deterministic to chance.

Obviously, the probability array size grows as the square of the previous exponentially growing outcome space. For all but trivial problems this conversion can cause serious memory limit problems especially in Markov decision process models where a chain of deterministic nodes can require astronomical amounts of storage space when converted to chance nodes. The software packaged developed for this thesis research includes several methods of limiting the size problem associated with deterministic node processing (See Chapter 3).

2.4 Additions Needed for Dynamic Programming

We have seen that solving an influence diagram involves removing nodes through a series of expectations and maximizations which preserve the maximum expected value of the problem and reveal the optimal decision policies. In traditional dynamic programming, the computational efficiencies come from the fact that each part or stage of a problem can be separated out and solved as a sub-problem independently from the other stages. When the individual stage solutions are later combined, the solution reached is the optimal solution for the problem as a whole. To gain the same advantages for influence diagram solution, the value function must be separa-

ble into parts and the expectation and maximization operators when applied to the separated value function components must still reveal the maximum expected value and optimal decision policies. While other types of functions may be separable, for the purposes of this research, a separable function was defined to be either a sum or product.

Tatman provides a proof, not duplicated here, that the maximization and expectation operators, when applied to separated value functions, do indeed reach the optimal solution [8:32- 50]. In order to represent the separated components of the value function a new node type was introduced: the subvalue node [8:26]. Figure 4 shows an influence diagram before and after separation of the value node into subvalue nodes. The numbers in parentheses near each node are the number of outcomes that node has. Notice, how even for this simple problem separating the value function reduces the size and complexity of the problem.

To incorporate these additions to the influence diagram language a new solving algorithm is needed. Like the previous algorithm it will always reduce an influence diagram to the original value node thus producing the maximum expected value and optimal decision policies. The separable value function influence diagram solving algorithm as given in Tatman and Shachter [7] is:

DEFINE PROCEDURE SEP-ID-EVAL AS
BEGIN
  check for oriented, regular diagram
  add "no forgetting" arcs
  eliminate all barren nodes
  split the value node into subvalue nodes if its function is
  separable and continue to split the subvalue nodes until a
  non-separable function is assigned.
  WHILE predecessors of the value node $\neq$ none DO

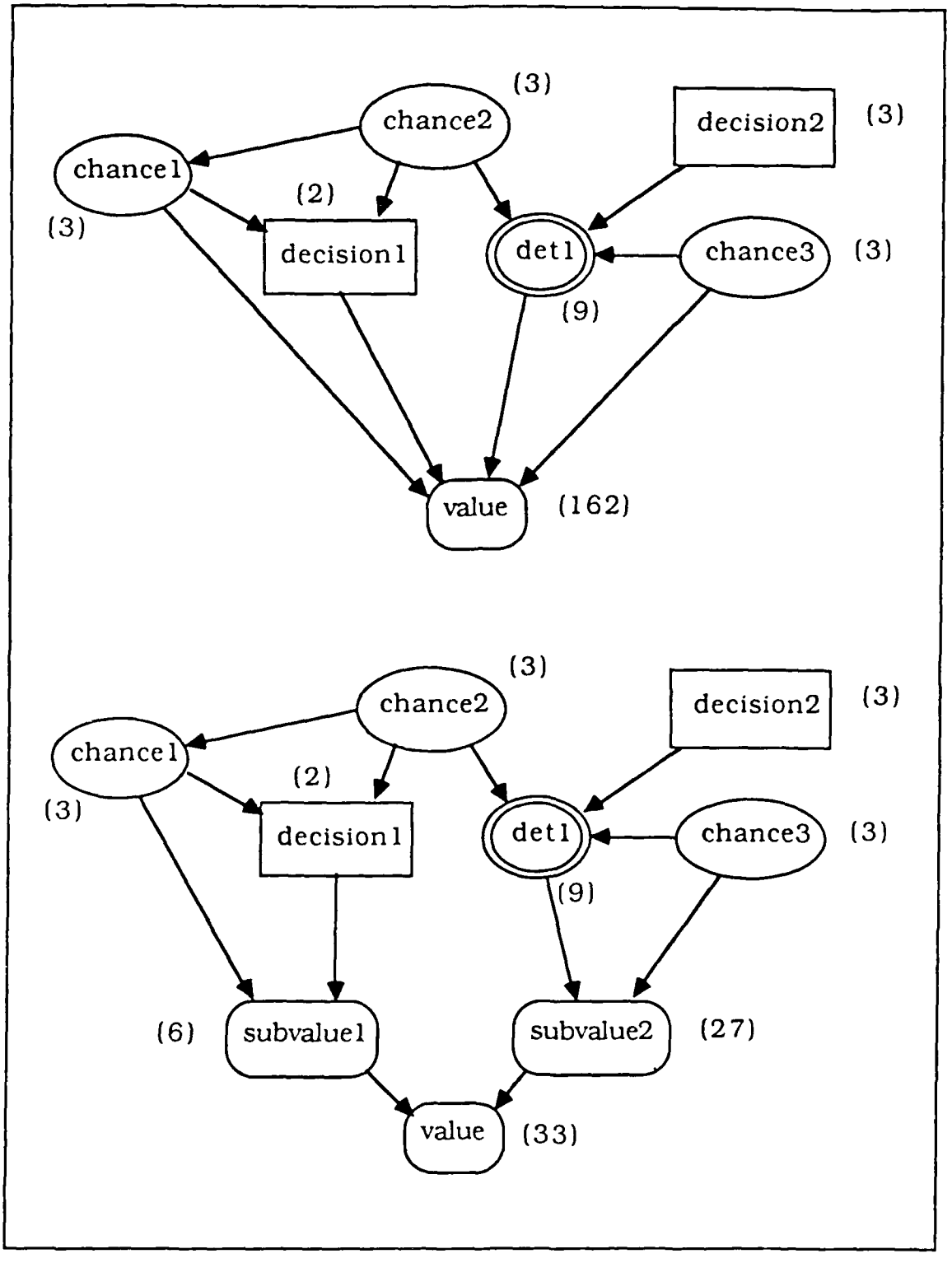

Figure 4. Influence Diagram with Separable Value Function

```
BEGIN
  IF any set of subvalue nodes predecessors are a subset of the
  predecessors of another subvalue node
  THEN combine the outcomes of the subset node's outcomes into the superset
  node's outcomes according the the function of the successor node.
  The subset subvalue nodes are eliminated.
  ELSE IF a decision node may be removed by maximization
  THEN BEGIN
    remove the decision node
    remove any created barren nodes
  END
  ELSE IF a chance node may be removed by expectation
  THEN remove the node by expectation
  ELSE BEGIN
    find a chance node which has the value node as a successor and
    only other chance nodes as its remaining successors
    WHILE chance successors exist DO
    BEGIN
      find a successor with no other path to the node
      reverse the arc between them
    END
    remove chance the chance node by expectation
  END
 END
END
```

Note, if the value function is not separable this algorithm will perform as the original to solve the diagram.

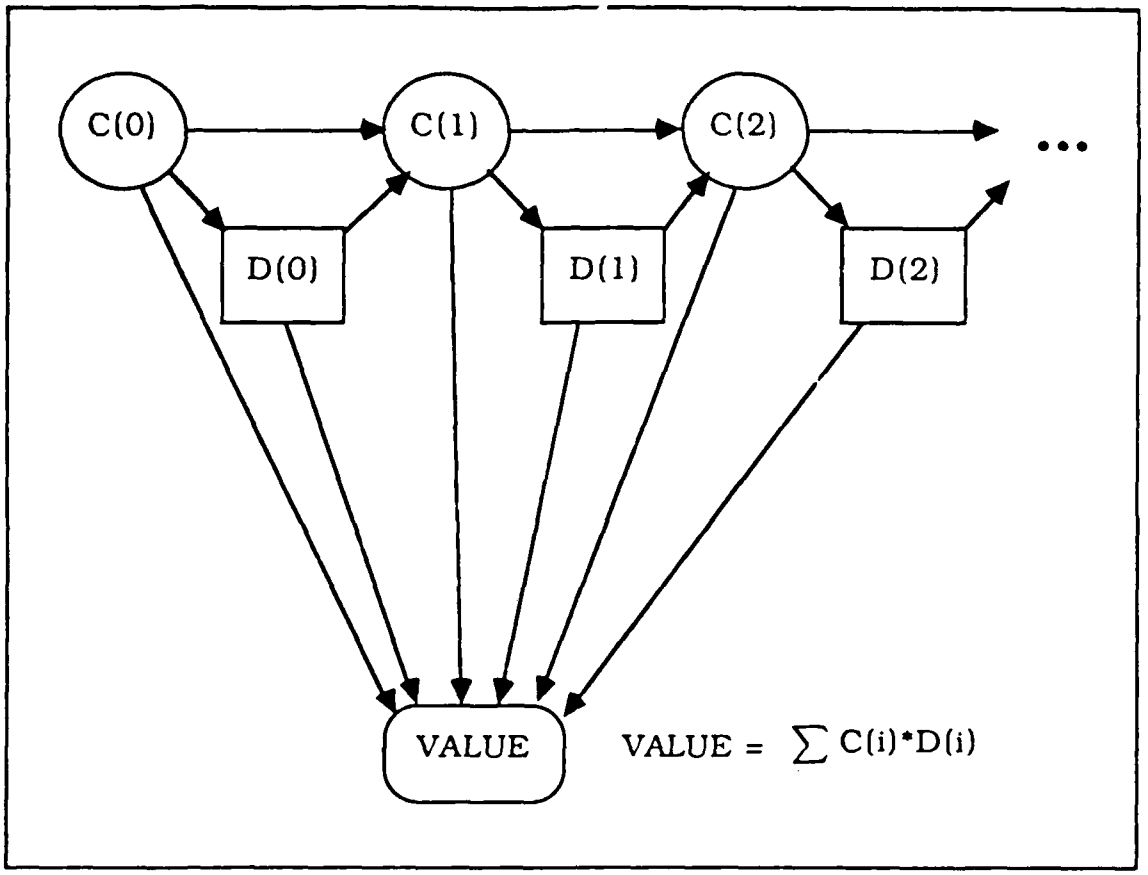

Figure 5. Influence Diagram with Variable Number of Stages

Now let us look a little more closely at exactly how much the subvalue node and a separable value function will save us in both storage space and computational effort. Figure 5 shows a typical Markov decision process with n stages consisting of a chance and a decision node with C_i and D_i outcomes respectively. For the unseparated value function the size of the problem will be:

$$\prod_{i=1}^{n} C_i D_i + \sum_{i=1}^{n} C_i + D_i$$

So for a problem with five stages and $C_i = 3$ and $D_i = 3$ the storage requirement for the just the node outcomes will be 59079 storage units (typically real numbers). The space required to store the probability arrays for the chance nodes is not included because it is not effected by the value node structure. For the separated value function there will be five subvalue nodes each with one chance and one decision

predecessor. The size of the outcome space for the problem formulated this way is:

$$\sum_{i=1}^{n} C_i D_i + \sum_{i=1}^{n} C_i + D_i + 1$$

For the example above the storage space required will be only 76 storage units or $\simeq 0.1\%$ of the above. The number of computations required to reach the solution is similarly reduced for the separated over the non-separated case.

This chapter has defined the influence diagram basics and the additions needed to perform dynamic programming like operations on separable value functions. The next chapter will discuss the implementation of these procedures in software.

III. The AFIT Influence Diagram System (AFids)

This chapter describes the software package developed in conjunction with this thesis research. The AFids software addresses the shortcomings of the other influence diagram processing software available and provides many influence diagram features in a user friendly, fast, interactive package. The software was developed to meet several needs and to provide an accessible future research vehicle.

3.1 System Requirements and Goals

The major goals for AFids were:

- The primary goal of any interactive software package should be user accessibility. The user should be able to easily and naturally interact with the software to accomplish his tasks. Also important is the responsiveness of the system. Rapid response to user commands is critical to user productivity. With a system that responds rapidly it is easy for the user to maintain his concentration and he will be encouraged to experiment and ask the 'what if' questions which lead to increased insight. All these factors fall with in the over used term 'user friendly'. Thus, the attributes of user-friendliness were a major concern during the development of the software and were the driver behind several design choices.
- Computational efficiency was another major driver in the software development cycle. Speed of computation was required for the system to be usable for more than trivial problems. Real world problems are large. To handle them, the software must be able to process rapidly enough that real problems can be solved in a reasonable amount of time.
- As a research tool it was important that AFIT have complete control of the program from source code to unlimited distribution and modification rights.

AFids represents a starting point upon which later advances may be build. AFIT often provides thesis students to work on problems for the Air Force and other government agencies. Thus, the software should be available for AFIT to develop models with, modify as needed, and then deliver to the customer for use at their site. Alternatively the entire source code could be made available to a customer for modification by them.

- In order for the software to applicable to other research or thesis projects it should be written in a common high level programming language. The use of a modular program design would allow sections of the code to applied to other projects easily. Further, a high level language would minimize the problems with porting the software to systems other than the development system.

3.2 Implementation

This section will discuss some of the design decisions made in the development process, hardware and software used for development, and the requirements to run the AFids software package.

The hardware used for development was a Zenith Z-248 microcomputer equipped with a 20 Mb fixed disk and an EGA color graphics adapter and monitor. Besides being commonly available to government users the MS-DOS (like the Z-248) type of microcomputer is the type most commonly owned by AFIT students as well. Although other, superior, microcomputer systems were available, the decision to develop the software on the Z-248 was driven by the need for governmental customers and later generations of students to have easy access to the compiled program and source code.

The programming language used to develop the AFids package was Turbo Pascal 4.0 from Borland International. Turbo Pascal is a low cost high performance dialect of the standard Pascal high level programming language. It is very common

on MS-DOS microcomputers and is taught in many colleges and schools including AFIT. Turbo Pascal was chosen for the development language for several reasons:

1. The language includes a highly productive integrated development environment including editor, compiler, linker, and utilities. The language is easy to learn and produces fast, compact code.
2. Programs produced with Turbo Pascal are completely owned by the author, no royalties or licenses are involved.
3. Turbo Pascal produces compiled programs that are 'stand alone', requiring no support routines or libraries to run.
4. A set of graphics routines supporting all of the common graphic display adapters for the MS-DOS computers is included standard with the package, easing graphic programming work.
5. Turbo Pascal is a compiled high level language and meets the requirements for an interactive software package as discussed in the previous section.
6. Turbo Pascal is perhaps the most popular compiled programming language available for MS-DOS computers. Thus, any future researchers desiring to modify AFids will have the highest probability of being familiar with it, facilitating future work with the AFids source code.

While Turbo Pascal is not standard pascal, converting the AFids software package to other implementations of Pascal or Modula-2 (a language whose 'module' structures are actually closer to the Turbo Pascal 'unit' structures used in AFids than standard Pascal structures) will be eased by the modular nature of Turbo Pascal. The majority of the mathematical routines are standard pascal and only the user interface and display routines would need to be rewritten.

The program was written utilizing Turbo Pascal's 'unit' structure extensively. With this method of organization, procedures and functions can be grouped by functional area and level of operation. For example, all of the high level node functions

such as *Reverse Arc* and *Maximization* are in a unit while lower level operations such as *draw graph* and *menu- choice* are in other units. This structure will allow future researchers to extract the program elements from the AFids system at whatever level is needed. Similarly, all of the system dependent routines are grouped in a few units, the graphics display routines are a good example. A programmer converting AFids to run on another computer can simply replace the graphics and text display routines with ones that are functionally similar and the program should run on the new machine, the rest of the code need not be changed.

The final version of AFids requires an MS-DOS computer running MS-DOS version 2.0 or higher, at least 256K of RAM, at least 1 floppy disk drive, and an EGA color graphics adapter and monitor. Converting the program to other graphic adapters would be very easy as the graphics routines in Turbo Pascal are device independent but several variables in the AFids program assume an EGA size screen (650X350 pixels) and at least 4 colors. The program code occupies about 200K of memory leaving about 320K free for storage of nodes and their related data structures(with the standard 640K configuration). Assuming average size outcome spaces and probability arrays, 320K is enough for a several hundred nodes. A limitation of Turbo Pascal, that data objects occupy $\leq$ 65521 bytes of memory [10:337], limits the number of outcomes to about 5000 per node and limits probability arrays to about 11,000 elements. If the numeric coprocessor version of AFids is used these figures are reduced by about 25% because the real numbers the coprocessor uses require eight bytes of storage each as compared to six bytes for the non- coprocessor version.

3.3 User Interface Design

The user interface is always the critical link between the conceptually minded human operator and the extremely literal outlook of the computer. Rather than invent a whole user interface concept for AFids, the researcher borrowed several tried and true user interface systems from other popular microcomputer software.

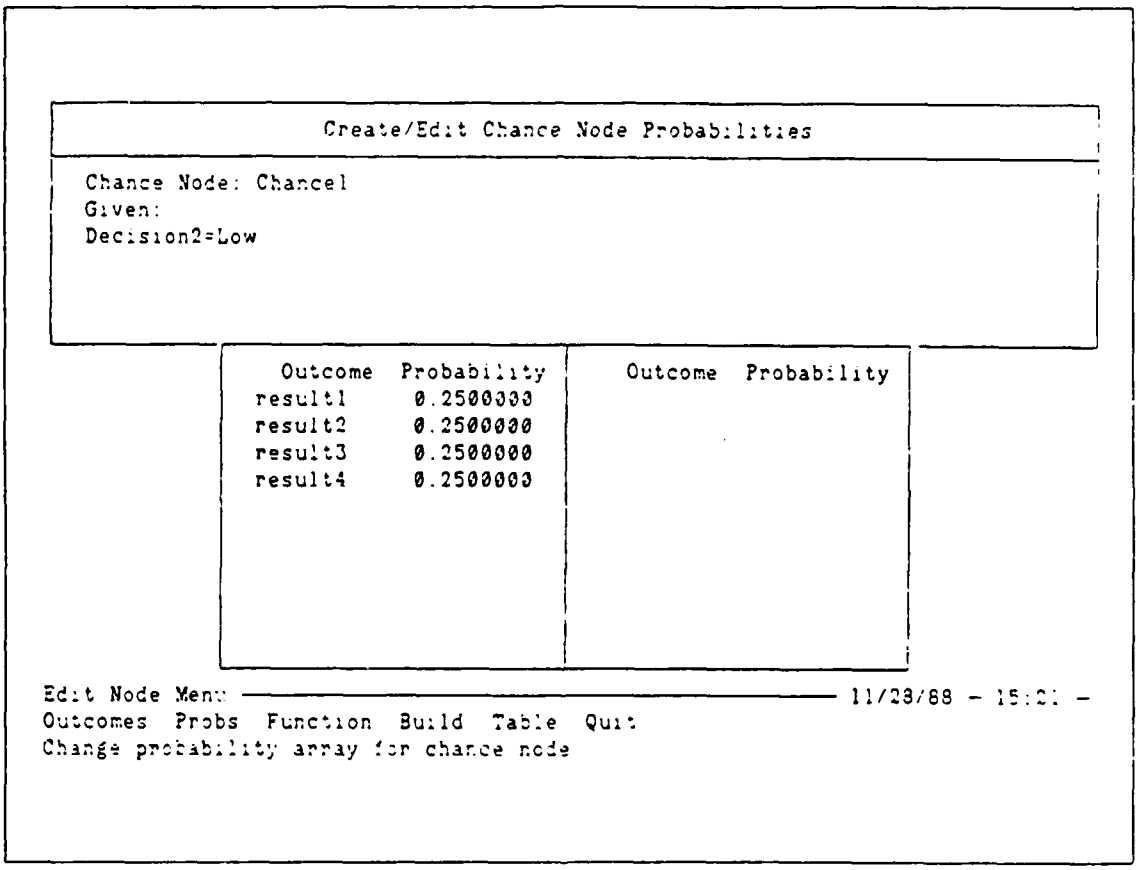

Figure 6. Typical AFids Text Display Screen

The primary user interaction with the program is through the keyboard. Figure 6 shows a typical display screen a user might see while using AFids. At the bottom of the screen are the menu lines the menu title, menu choices, and the explanation for the currently high-lighted menu choice. The user can navigate through the menus by either choosing the first letter of the menu choice or by high-lighting the menu choice and hitting the enter-key. This system is popular with programs such as Lotus 123. Moving through the menus with a system such as this provides both speed for the experienced user, who can change menus as quickly as he can hit the keys for the first letters of the menu choices, and explanation for the novice, who can read the mnemonic command names and get a further explanation by high-lighting a menu choice and reading the explanation line. Should the command selected require data entry, an entry screen such as the one shown if Figure 6 is presented. Each item of data needed has its own field for entry. Syntax, data type, and checking for values within a legal range are performed during entry. Sanity checks are also performed such as insuring that a probability distribution sums to 1.0. Feedback is supplied if an error is detected and the user is required to re-enter the data. A method for aborting any operation (usually the ESC key) in AFids is provided so that if a command is selected by mistake no permanent damage is done. Further, before irreversible changes are made to the diagram (such as solving it) the user is prompted to save the diagram to disk.

Figure 7 shows how the graphical representation of an influence diagram is displayed in AFids. The graph is displayed in exactly the same form as appears in influence diagram literature. Operations affecting the graphical structure of the diagram (such as adding or deleting a node or arc or manually choosing a node to remove) are carried out here. Node selection and placement are carried out by the point and click method. When an operation involving a node is selected the user is presented with the graphical display of the diagram he then points to the node or nodes involved with a graphics cursor and hits the enter-key. As in the data entry

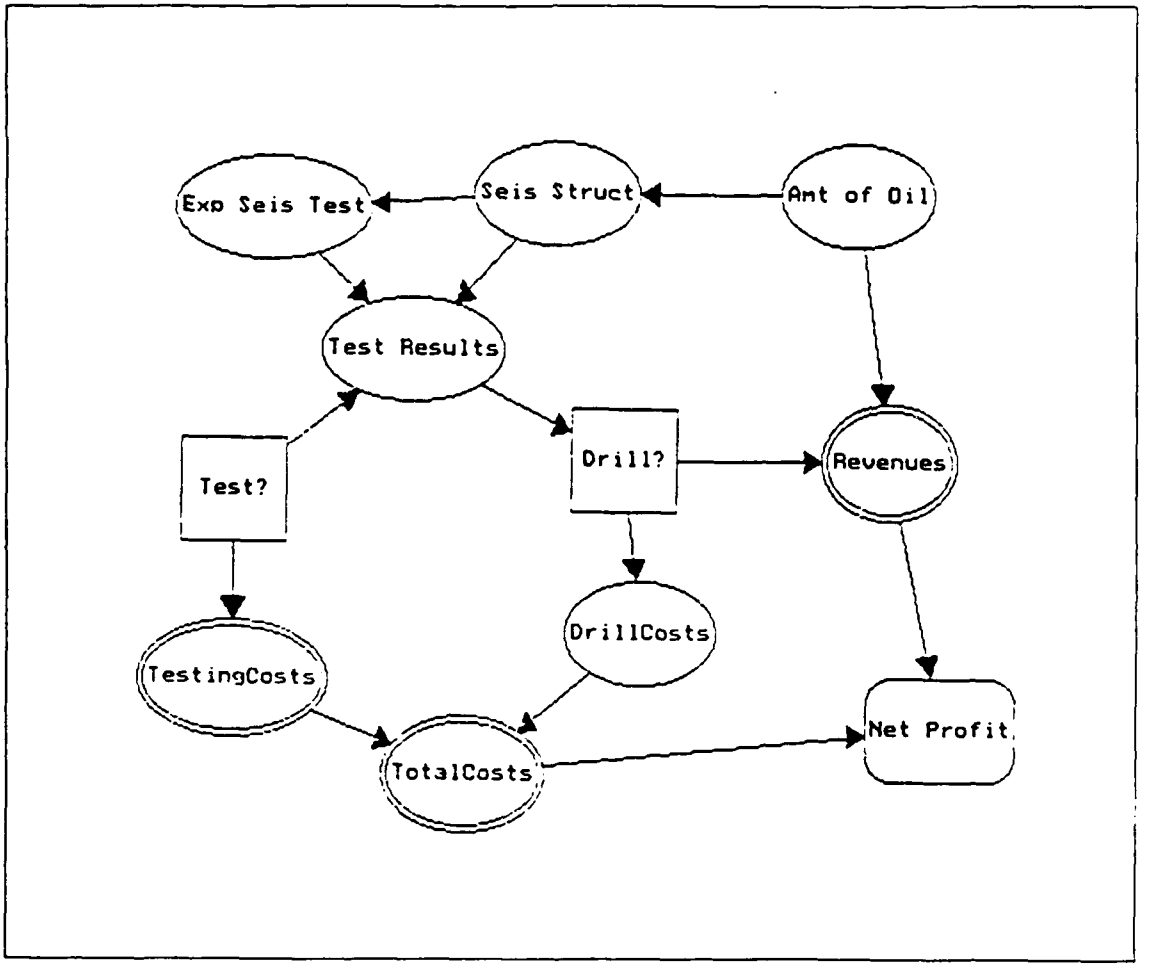

Figure 7. Typical AFids Graphics Display Screen

case the user can always abort any operation without causing any damage. See Appendix A for full details of the AFids user interface and command set.

Error messages are provided for all system detected errors. An English language explanation accompanies each error message for the convenience of the user.

3.4 Program Philosophy and Data Structures

This sections describes the structure of the AFids code as it applies to representing the influence diagram and some of the programming philosophy the researcher tried to include in the program design.

The Pascal heap feature provides a very convenient way of dealing with data storage requirements which cannot be determined at compile time. The heap represents all of the available memory in the computer not taken up by AFids and other programs or data. A program can reserve memory from the heap and return it when it is no longer needed. AFids uses this feature extensively. Only the bare minimum of program variables and data structures are declared at compile time. The vast majority are allocated from the heap on-demand at run time. The purpose of this is to avoid as much as possible any arbitrary limits on the diagram such as a limit on the number of nodes in the diagram, or on how many outcomes a node may have. The only limits are those on the maximum size of data objects mentioned above and that limitation is a function of the microprocessor used in the MS-DOS type microcomputers. Linked lists provide the method by which the diagram can be built using data objects allocated and de-allocated at will during program execution. Figure 8 shows how data elements in the influence diagram are stored using linked lists. Virtually everything in AFids is either a member of a linked list or an element of a node in a linked list. The other major data storage object is the linear array. Since the size of objects like probability arrays and outcome arrays cannot be determined until the influence diagram is being constructed, linear arrays of the exact to hold all the required elements are allocated from the heap at run time. In the case

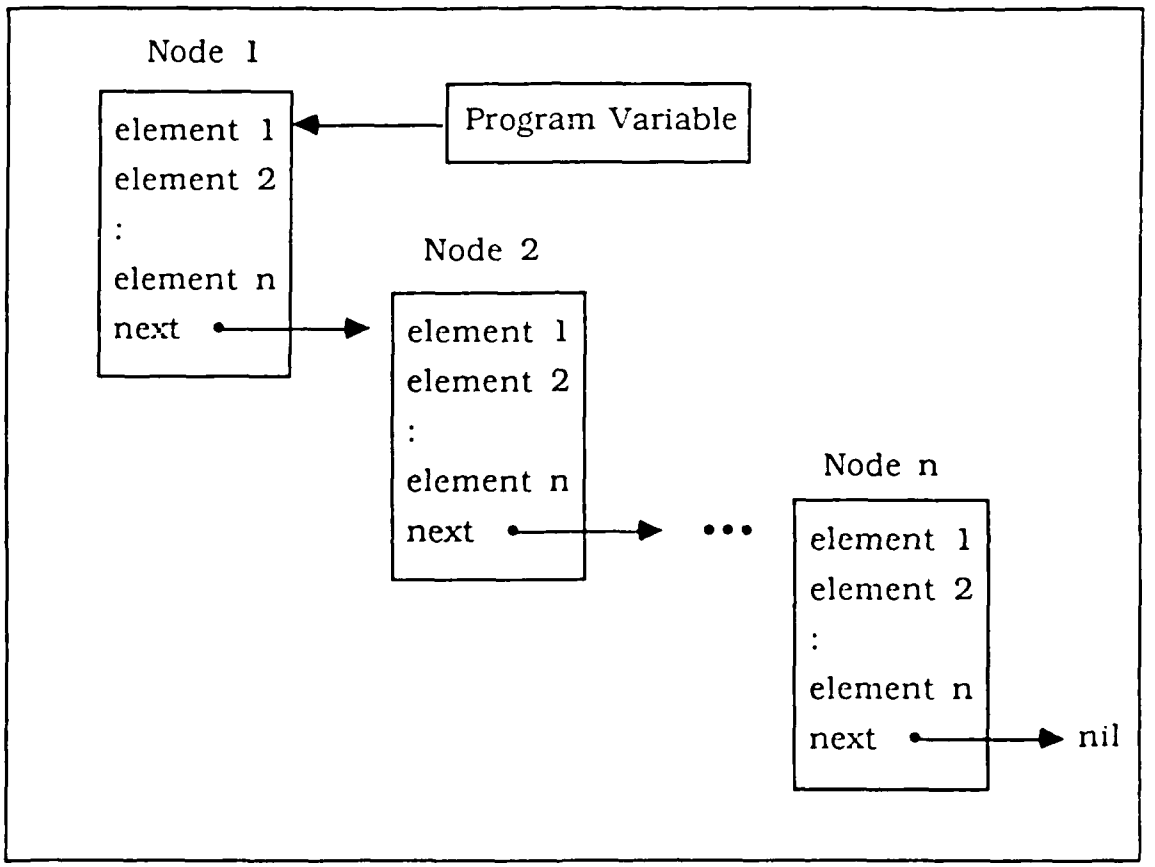

Figure 8. A Linked List Data Structure

of probability arrays the number of rows and columns must also be stored for the program to access the array elements correctly. Using this method allows nodes to occupy exactly as much space as they need and avoids all limits on the diagram and its components except for the maximum size already discussed. Figure 9 shows the components of a influence diagram node as used by the AFids system. Note, only those elements needed for a particular node type are actually used. Elements that do not apply to a a certain type of node such as a probability distribution for a decision node are either set to nil or to 0. As nodes are created during the course of a session with AFids a chunk of memory the size of the base node structure is allocated from the heap. Then as processing of the diagram continues various components such as the outcome array are added, also being allocated from the heap.

3.5 AFids and Dynamic Programming

To implement dynamic programming techniques AFids essentially uses the method outlined in Tatman and Shachter [7:20-25] by creating subvalue nodes if possible by separating the value node's function. Special functions have been defined in the function calculation procedure to help AFids determine separability. The two functions *SUM()* and *PROD()* are known to be separable by AFids. The arguments for the functions are an arbitrary length list of functions of the various predecessor nodes. One subvalue node is created for each term in the function argument and has assigned as its function the term in the original function. The new node's predecessors are those nodes whose names are mentioned explicitly in its function. AFids will then split subvalue nodes if their function is one of the separable functions. The diagram is then solved using the separable value function solution algorithm as given in Chapter 2.

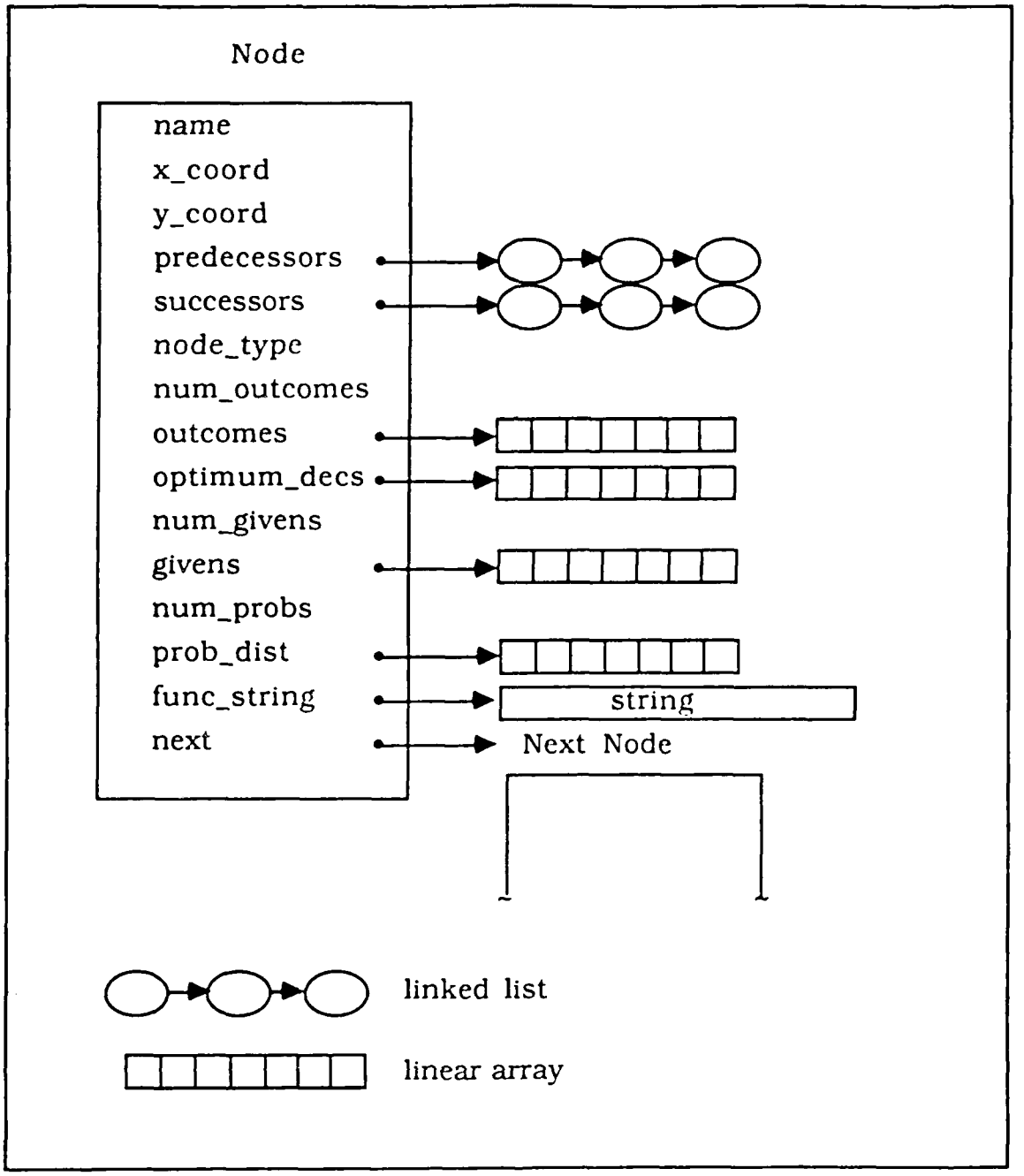

Figure 9. The AFids Data Structure For An Influence Diagram Node

3.6 AFids Unique Features

The AFids software package was design from the ground up to accommodate the needs of separable value functions and dynamic programming. As such it incorporates several features not found in other influence diagram processing software. Extensive features for the handling of deterministic nodes are included. The existing influence diagram software handles deterministic nodes by simply converting them to chance nodes before the solving process begins. As we have seen this is not practical for diagrams containing chains of deterministic nodes as are common in Markov decision processes. AFids handles the exponential growth of the outcome space in these chains in four ways.

1. When AFids converts a deterministic node into a chance node it combines outcomes which have the same numeric value and only creates one entry in the outcome table to reduce the size of the outcome space and the probability array. In the default case the 'same numeric value' is defined to be within 1.0 $\times 10^{-8}$ this small value will match numbers with only round-off error differences such as 1.99999999 and 2.0.

2. AFids introduces the concept of a round-off value for a deterministic node function. The round-off value is optionally set when the user enters the function. At the time the deterministic node's outcomes are calculated they are rounded to the nearest multiple of the round-off value. For example, if the round-off value is set to 3.0 the only outcomes the node could have would be ..., -3, 0, 3, 6, 9, ... Thus, the user has control over the precision used when calculating the outcomes of the node. Obviously this can help the exponential outcome growth problem, when used in conjunction with item 1, but does not eliminate it because a node can still have any number of outcomes they are simply restricted to multiples of the round-off value.

3. AFids also introduces a concept called 'outcome limiting'. As implemented in AFids, a set number of outcomes can be defined for a node or group of nodes. When the outcomes of the function are calculated they are rounded into the required number of 'bins' whose values span the high and low outcome values calculated. Then when the node is converted to a chance node only the set number of outcomes are allowed. Thus for chains of deterministic nodes the outcomes space will not grow exponentially or even linearly it will be fixed.

4. AFids also introduces the concept of algebraic combination to eliminate deterministic nodes. Figure 10 illustrates the concept of algebraic combination. Essentially deterministic nodes are eliminated before their outcomes are calculated by algebraically combining their functions into the value node function. It is easy to see that algebraic combination cannot increase the size of a problem but may reduce it significantly. When processing a diagram for dynamic programming the procedure is to algebraically combine all deterministic nodes possible and then proceed to separate the value function if possible. That way the problem of converting deterministic nodes to chance nodes may be at least partially eliminated.

Another unique feature of AFids relating to deterministic nodes is the function calculation procedure. The available influence diagram software handles deterministic functions by defining them in-terms of LISP language functions which are very confusing to a user not familiar with LISP (DAVID has a limited capability for representing functions in algebraic format). AFids functions are entered as strings by the user. The format is the same as many common high level programming languages such as BASIC, FORTRAN or Pascal. Many predefined functions are included in the AFids package for scientific, financial, and logical calculations(See Appendix A for a complete list of AFids functions). Adding new functions simply requires adding the new function name to the list in AFids and providing a subroutine to do the

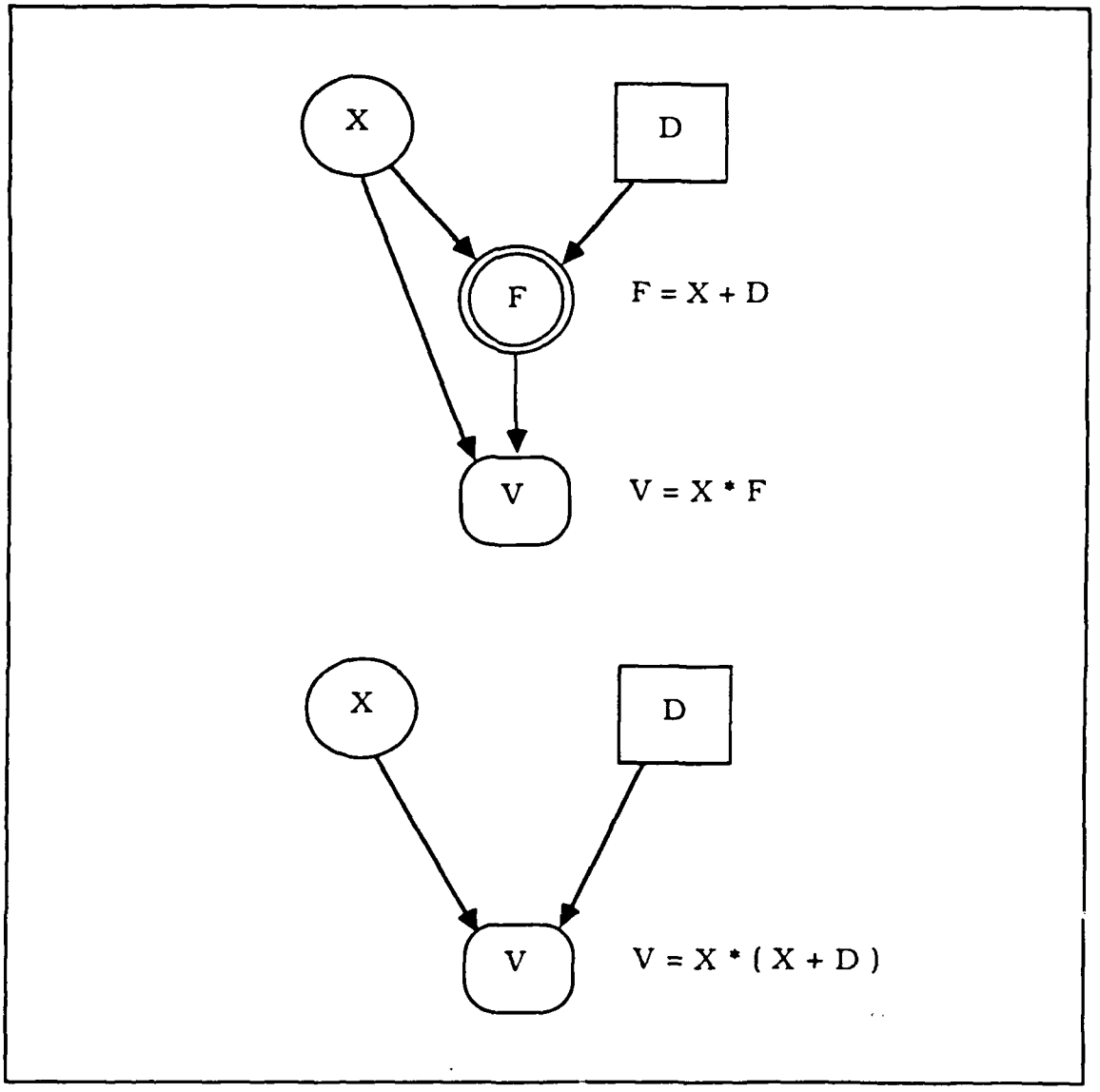

Figure 10. Functional Combination in Deterministic Nodes

calculation (AFids must be recompiled). This makes it easy for customized functions for a particular customer's problem to be added to AFids.

The final 'unique' feature of AFids is speed. The program processes diagrams very quickly. This is due partially to good program design and partially to the Turbo Pascal compiler. For example, AFids solves the oil wildcatter problem (diagram is given in Chapter 1) approximately 7 times as quickly as DAVID and about 20 times faster than PerForma.

IV. Applications

This chapter will discuss two sample applications of dynamic programming and influence diagrams to decision problems. Several insights into the suitability of influence diagrams to various types of problems are revealed. The first application involves a typical Markov decision problem of deciding when to send aircraft in for maintenance. The second involves allocation of resources between procurement and research and development for artillery shells. This second application, as formulated, has very little uncertainty in it and would traditionally be solved with linear programming.

4.1 Application #1: Aircraft Maintenance

In this application a maintenance officer for an aircraft squadron wants the optimal policy for when to send his aircraft to the depot for maintenance. The aircraft are rated to be in one of four readiness categories, which he determines once each month. At that time he can decide to send the aircraft in for depot maintenance or to leave it in service until the next month.

Figure 11 shows the structure of the problem as formulated in an influence diagram. The nodes labeled S_n represent the random state of the aircraft in month n. The nodes labeled D_n represent the maintenance officer's decision for that month whether or not to send the aircraft for maintenance (month 0 is the initial state of the aircraft). The node labeled S_5 represents the aircraft state at the end of the fifth month (a salvage value). Notice that the arcs in the diagram show that the officer knows the state of the aircraft in month n before he makes his decision and the state at any time period depends on the previous state and decision. There is an availability cost associated with aircraft being in less than top shape due to sorties missed, extra crew maintenance etc. There is also a cost for sending the aircraft to

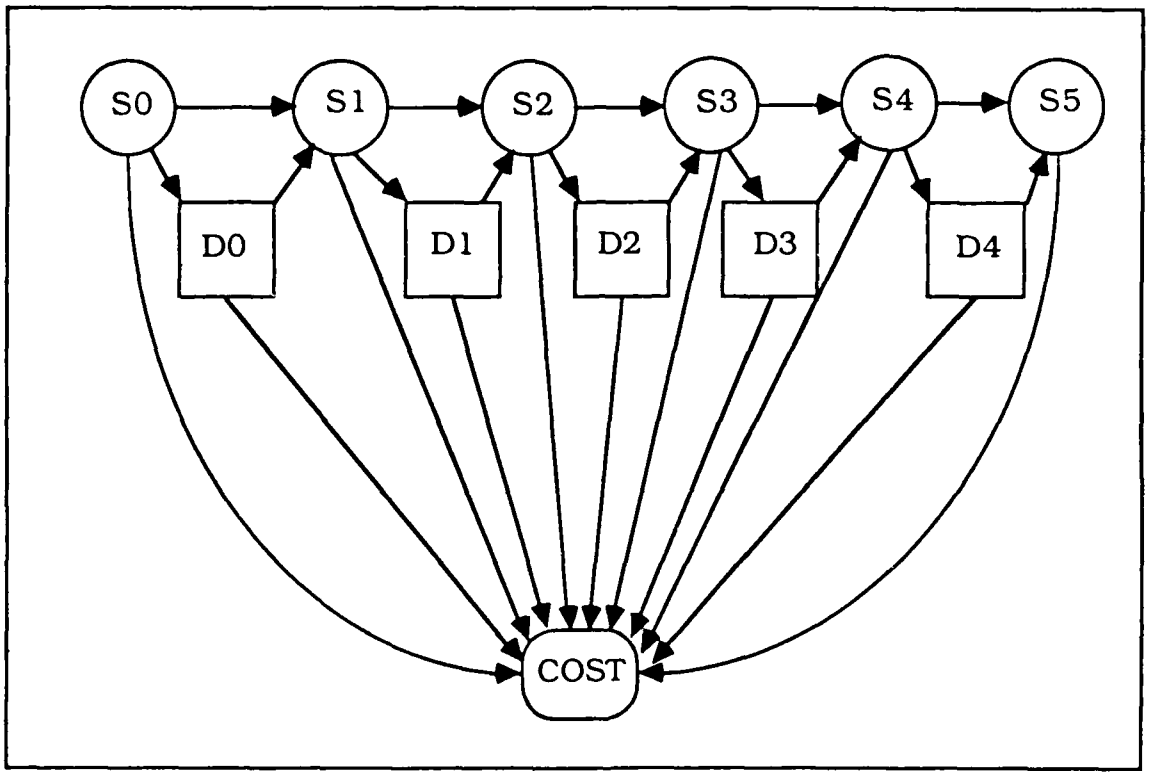

Figure 11. Influence Diagram for the Aircraft maintenance Problem

the depot. The total cost will then be:

$$\sum_{i=1}^{n} Cost_{availability_i} + Cost_{depot_i}$$

What the officer would like to do then, is minimize his cost for operating the squadron. He must trade off between availability and maintenance costs. The outcomes of the aircraft state and maintenance decision influence the value node, which represents the cost of operating the squadron for the given time period. In the internal influence diagram storage the costs would be stored as negative numbers. Thus, when the expected value is maximized the cost is driven toward zero.

When solving this diagram, notice that for each stage(month) there are eight possible combinations of outcomes for the two nodes composing that stage(four aircraft states and two decision states). Thus, the total number of outcome combinations for the value node is $8^5 \times 4$ or 131,072. Storing this many numbers exceeds the capacity of AFids, in fact at 6 bytes per real number that many numbers would require $\simeq$800K of memory. The time required to calculate the values would also be prohibitive even if the results could be stored, especially for an interactive system (AFids, which is the quickest influence diagram solver, would require over five hours just to calculate the value node outcomes, not to mention doing the expectations and maximizations to actually arrive at a solution).

Since the value node is a simple sum of the stage values, it fits the criteria, defined in Chapter 2, for being separable. The value node will be split into six subvalue nodes each representing the cost for a stage (the stage reward, in dynamic programming terms). Figure 12 shows the influence diagram with subvalue nodes added. The expectation and maximization operations to solve the diagram may now be carried out with respect to the subvalue nodes according to the solution algorithm given in Chapter 2. The number of outcomes for all of the subvalue nodes is $(8 \times 5)+4$ or 44 and AFids solves the diagram in about three seconds. The steps followed are exactly the same as would be taken if the problem was solved by classical dynamic programming. The power of influence diagrams in solving this type of problem lies

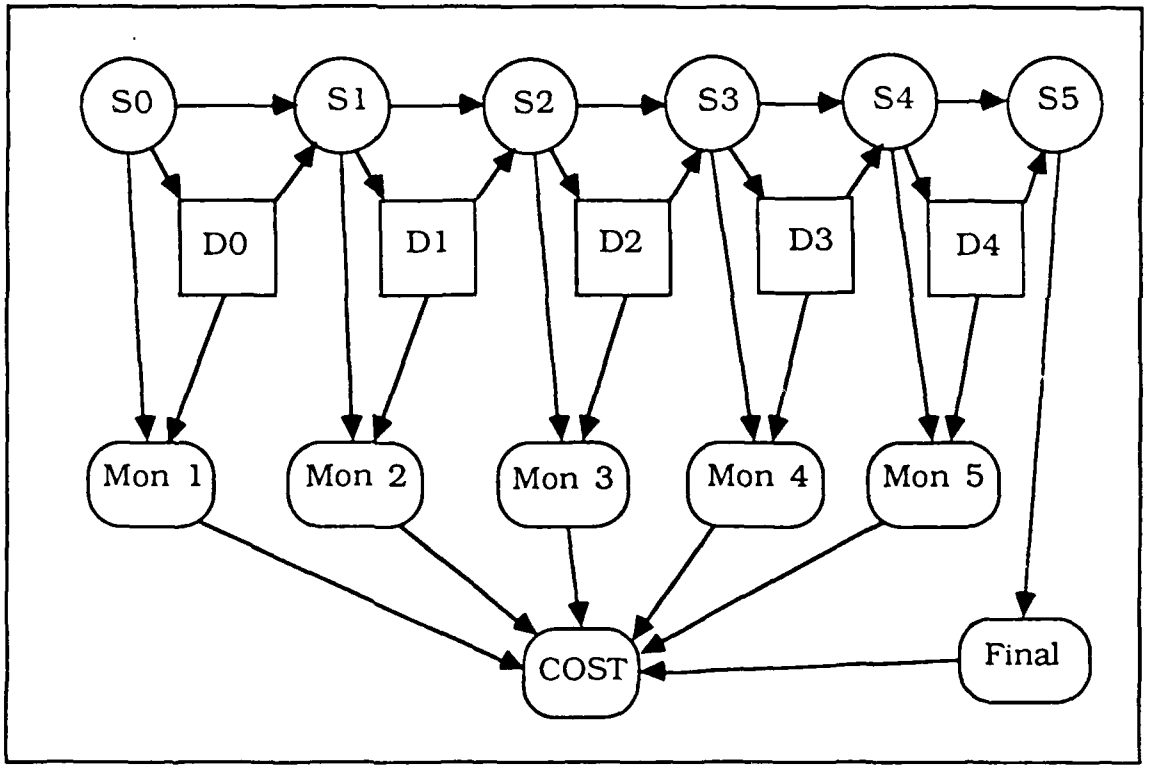

Figure 12. Influence Diagram for the Aircraft Maintenance Problem with Subvalue Nodes Added

not only in providing a solution to the basic case but quick and efficient solution of variations on the basic case. Addition of time lag dependencies and calculation of the value of information and control are easy with influence diagrams (and very difficult to calculate using dynamic programming or classical probability theory) and a good software package such as AFids. Much insight into the true relationships among the various variables can be gained.

4.2 Application #2: Procurement vs R&D

In this application a decision maker is trying to decide how to allocate his resources for an artillery shell program. He can either spend all of his budget allocation on procurement or he can spend 25% of his budget on research and development to improve the effectiveness of the shells. His objective is to maximize the number of targets the stockpile of shells can kill in each year of a five year program. The target killing potential at the end of each year provides a convenient stage reward with the final objective function being the sum of the yearly values. Figure 13 shows the components of a yearly stage and their relationships with the previous and following years. The nodes labeled N_k represent the number of weapons required to kill one target. Nodes labeled *Imp* represent the improvement in N_k resulting from a decision *R&D* to 'buy' R&D for that year or not. Finally, nodes n represent the number of weapons in the inventory at the end of the year. The number of weapons is influenced by both the budget node B and the decision. Notice that the decision maker knows the N_k and n for the previous year at the time he makes his decision. There is very little uncertainty in this problem compared to the previous example (only four out of twenty two non-value nodes are chance nodes vs six out of eleven in the maintenance problem).

Figure 14 shows the complete influence diagram for the procurement problem with the subvalue nodes already split out from the value node. When attempting to solve this diagram we immediately run into problems with the large number

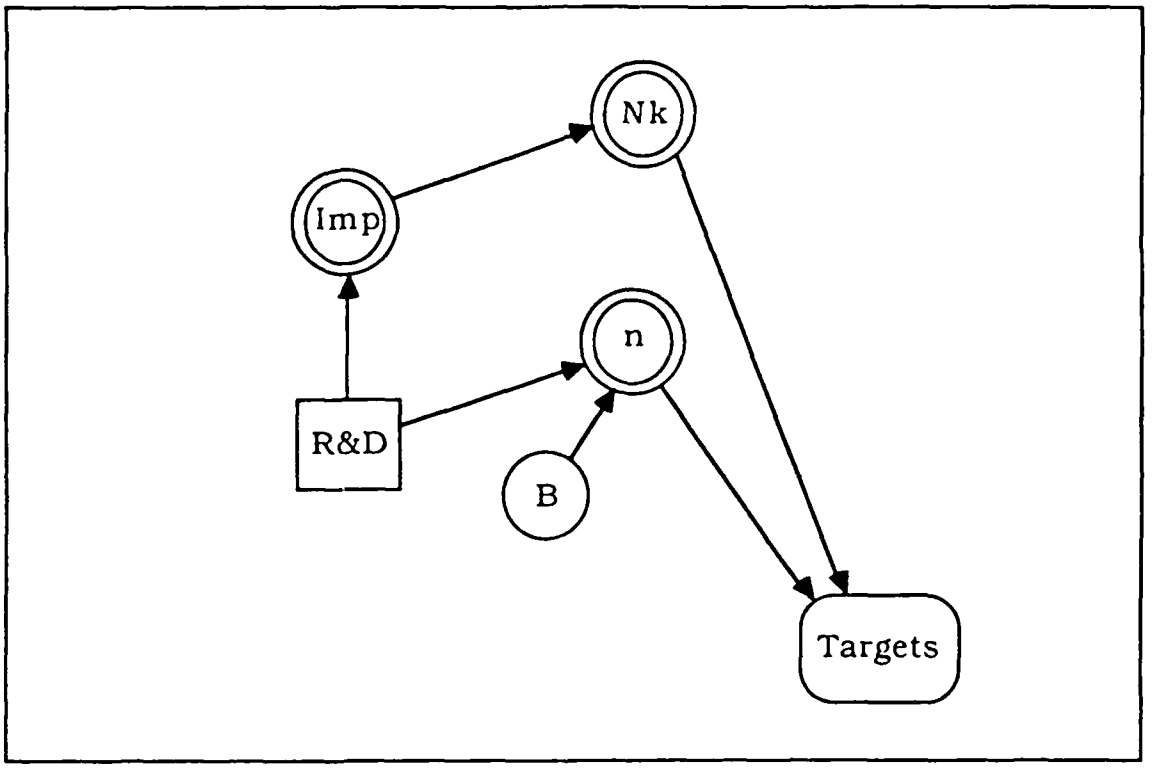

Figure 13. A Stage in the Artillery Shell Problem

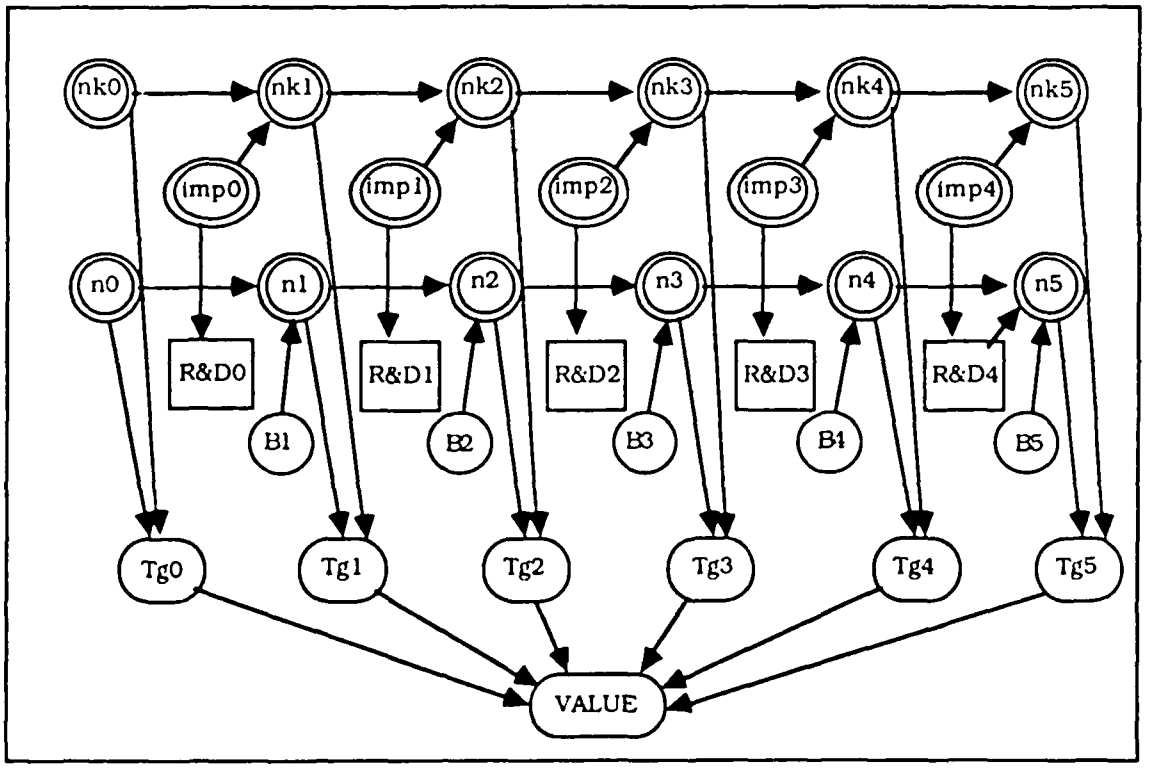

Figure 14. Influence Diagram for the Procurement Problem

of deterministic nodes. According to the algorithm, before solution can begin all deterministic nodes must be converted to chance nodes. We can see that both the N_{ki} and the n_i nodes form a chain in which each year is influenced by the previous year's outcomes. To illustrate the problem let us calculate the number of outcomes node n_4 will have assuming three different possibilities for budget level (B_i), and the two possible outcomes for the decision node $R\&D_i$. For year i the size of the outcome space will be: $2 \times 3 \times n_{i-1}$. Thus for year four there will be 1296 outcomes. The maximum size of the probability array will be 1296^2 or 1,679,616 elements. This number is obviously far beyond the capability of existing software, and this is but a small simplified problem. Clearly these chains of nodes which have the number of outcomes depending on the number of outcomes of their predecessors are a problem.

AFids provides two ways of dealing with this problem. First, the outcomes can be rounded to some precision (for example to the nearest 100). AFids will then recognize the identical outcome values and only create one entry in the outcome list for that value. This method obviously just delays the exponential explosion. Still it may be enough to get a solution for small problems such as this. For larger or more complex problems the only solution is to make the number of a node's outcomes independent of the number of outcomes its predecessors have. AFids is unique among influence diagram software in implementing 'outcome limiting' for nodes. When the problem is formulated the number of outcomes a deterministic node may assume is set. Then, when the node is converted to a chance node *all* the outcomes are calculated and the range between the high and low values is determined. The values for the set number of outcomes are created by dividing the range into that many segments. Each calculated outcome is then rounded to the closest segment value. This method effectively blocks the exponential growth problem because a node can only have up to some fixed number of outcomes. For example if the n_i nodes were limited to ten outcomes, the maximum number of outcomes node n_4 could have would be $2 \times 3 \times 60$(max number of outcomes for the previous node n_3) or 360

which would then be reduced to $2 \times 3 \times 10$ or 60, and the resulting probability distribution would have at most 3600 elements. This will be true for a chain of any length so long as the outcomes are limited. The method of letting the un-rounded outcomes determine the values for the limited outcomes, rather than choosing the actual values during formulation, preserves the maximum range in the outcomes and frees the modeler from trying to estimate the outcome range beforehand. Once this problem is solved the diagram can be solved using the solution algorithm. As in the previous example, the subvalue nodes keep the outcome space of the value node from becoming prohibitively large.

A second set of problems with an influence diagram solution to this problem is not so easy to solve. The problems are with the simplifications that must be made to the problem in order to formulate it as a reasonable influence diagram. The example shown in Figure 14 has 28 nodes and makes one yes or no decision concerning R&D for one weapon for five years. The original problem from which this example was drawn was to allocate a large budget amongst 16 different weapons, each of which would then allocate its budget, as in the example, between procurement and R&D over a 10 year period. Several additional objectives from the original problem were not included, such as the weapons attacking different types of targets at different ranges, and limits on the maximum and minimum numbers of weapons of each type that could be procured, a varying amount of money spent on R&D, constraints on program increases or decreases in adjacent years, etc. An influence diagram of the entire original problem would have thousands of nodes and take a very long time to solve. The linear programming formulation, on the other hand, consists of about 1,000 decision variables, a few hundred constraints, and can be solved in a few minutes with a microcomputer based LP solver. The LP solution also allows (almost requires) continuous variables which the influence diagram does not. One shortcoming of an LP solution is that the uncertainty in the problem must be effectively ignored. So it would seem an influence diagram system with continuous

variables to represent the functional relationships as well as handle the uncertainty in the standard manner could do a better job than LP at modeling this type of problem.

4.3 Application Observations

In the first application example influence diagrams could be used effectively to answer the primary question, and to provide efficient handling of variations on the basic problem. The nature of the solution process can reveal many insights into the underlying relationships among the problem variables and influence diagram software provides the vehicle for obtaining the information in quickly and in a manner easily understood by decision makers and modelers alike. The use of the subvalue node along with software to recognize the separability of the value function allows problems of this type, which were formerly computationally intractable, to be solved efficiently. In this first example, the primary question to be answered was to find the best policy for sending the aircraft for maintenance. Random variables played a large part in the problem (over 50% of the nodes were chance nodes). Influence diagrams have always been good at solving decision problems containing significant uncertainty. This application seems to be an example of a problem that can be solved well by influence diagrams especially when the separability of the value function and subvalue nodes are used.

The second application example essentially tried to force influence diagrams to do the job of linear programming. The primary question to be answered was whether or not to pay for R&D on the weapon. However, the large number of simplifications and discretizations necessary to fit the problem into an influence diagram formulation shed considerable doubt on the validity of the solution even before the solution was attempted. Most of the computational effort was spent on converting functional relationships into a form which the influence diagram could manipulate. A special operation was needed (outcome limiting) to enable influence diagrams to handle

the combinatorial explosion of the deterministic node chains, and the rounding of calculated values decreased the precision of the answer. Further, while theoretically able, influence diagrams were not practically able to handle the full problem and certainly not any complications to the basic case, which in this type of problem usually take the form of additional variables or constraints. The problem contained little uncertainty, the majority of the information in the model was represented by the functional expressions of the N_k, Imp, and the n nodes. This example showed that influence diagrams, especially with the features of the AFids software package, can be used to solve many types of problems. However, this example would seem to be one which does not lend itself well to solution via influence diagrams.

V. Areas for Further Research and Conclusions

5.1 Areas for Further Research

The AFids software package developed in conjunction with this thesis research provides capability equal, in most cases, to the state of the art in commercial microcomputer influence diagram software and incorporates several unique features. Future enhancements to the software may take one or both of two directions.

1. The software may be polished and features added with the intention of producing a commercial quality package. The software could then be distributed as a general problem solving, decision analysis aid to customers throughout the US government. AFids could become a government standard for influence diagram processing. AFIT would, of course, have to be willing to support the software at a commensurate level.
2. The software could be extended as a research vehicle for examining and implementing techniques and routines on the cutting edge of influence diagram theory. In this capacity it would be suitable for faculty or student research, for class work, and for 'one time' problem solving for customers. AFIT would not have to support the package in this capacity.

Following is a list of additions to AFids the researcher feels would extend the usefulness of the program. The suggestions are in rough order of importance.

- The addition of deterministic and stochastic sensitivity analysis would make AFids equal to the best in commercial influence diagram software. The techniques for implementing this are well known but there was simply not enough time to include sensitivity analysis in the thesis version.

- The addition of explicit value lotteries would also enhance the completeness of AFids for general decision analysis work. As above, this addition requires nothing new in influence diagram theory, just programming effort.
- Continuous decision and random variables are not supported by current influence diagram software. Addition of this capability would advance the state of the art in influence diagram software and would allow exact solution of a large class of problems which can currently be handled only by approximation. Only the basic work in this area has been done [4], so implementation would require a research effort, analogous to the current thesis research, to implement the theory in a usable form.
- Another area of research which has not received much attention is the concept of an influence diagram node representing a vector of variables. This capability would allow influence diagrams to model larger problems without the visual complexity that explicitly representing each variable can create. This addition would require significantly more effort than the previously mentioned additions. Both extension of influence diagram theory and software implementation would be required.
- The ability to handle either values or probabilities as symbolic data would provide capability for analyzing problems where explicit values for certain quantities are difficult or impossible to obtain such as the dollar value of a human life. Implementation of symbolic data processing in influence diagram software would be a major advancement in the state of the art.
- Porting AFids to run on other computer systems would increase its availability to all types of users. A mainframe version would offer increased processing speed and eliminate the size limitations, possibly at the expense of interactive graphics. Other microcomputer versions would allow students and government customers with other makes of computer to use AFids.

- Following are a list of improvements the researcher feels would improve the current MS-DOS compatible version of AFids(in no particular order):
  - Modify AFids to run on different graphic display adapters.
  - Convert AFids to run completely in graphics mode. The menu and data entry displays could be implemented as windows.
  - Add on-line help support.
  - Add mouse support.
  - Improve the output reporting both printed and on-screen.
  - Support different printers and add plotter support (currently only Epson compatible printers are supported).
  - Modify the program logic to eliminate the size limitations imposed by Turbo Pascal, possibly switch to a different Pascal or other language such as Modula-2.
  - Allow nodes to represent the outcome of a separate node or diagram stored in a separate file. This creates essentially a third dimension to the diagram where the results of other analysis can be included the current model without laboriously re-entering data each time the other analysis is modified.
  - Add links to spreadsheet and database software.
  - Improve function calculations by improving the function location algorithm and by storing string numbers in internal real number format(these two improvements could easily reduce the function evaluation time by a factor of three).

5.2 Conclusions

The software package, AFids, developed in conjunction with this thesis research meets the goals set forth in Chapter 1. It is user friendly, interactive, and

provides a graphical environment for the processing of influence diagrams. The software is written in a popular high level compiled language providing both excellent performance and easy access for future researchers. The quality of the software is such that it is suitable for using during the entire decision analysis cycle from initial formulation to solution. It is cost free to all US government agencies and has no restriction, except for commercial use, on distribution of either the compiled program or its source code.

Several advancements to the state of the art in influence diagram software are incorporated in AFids. Automatic use of the separability of a value function increases solution efficiency without additional effort on the part of the user. Several types of deterministic node processing are implemented including automatic algebraic function combination, optional deterministic node value rounding, and optional deterministic node value limiting. No other influence diagram software, commercial or otherwise, includes these features.

The software has been applied to several application examples and demonstrates the validity and efficiency of the subvalue node and separation of the value function. The decision process, a type of problem previously computationally intractable for influence diagrams, has been added to the list of problem types solvable by influence diagrams. The application examples also showed that influence diagrams are ineffective in problems where there is little uncertainty or where a large portion of the problem information is contained in functional relations.

Appendix A. *AFids Users Manual*

A.1 Introduction

The AFIT Influence Diagram Sy·tem (AFids) is a general purpose influence diagram processing software package incorporating several unique features for processing diagrams via dynamic programming and for handling the special problems associated with deterministic node processing.

AFids is suitable for creating, editing, and solving influence diagrams of all sizes and complexity levels. The primary AFids input device is the keyboard. Commands are entered into a menu interface similar to that of Lotus 1-2-3. A graphical representation of the influence diagram is also presented and nodes are selected for operations via the point and click method. Data entry screens are provided when numeric or string data is required. Data checking for type, syntax, and value range is done on user entered data. Sanity checks are also made, such as making sure a probability distribution sums to 1.0. This provides an intuitive command system which provides the new user with mnemonic commands and explanations in menu selections and speed for the experienced user who can execute commands as quickly as he can hit the first letter of the command.

This users manual does not attempt to teach the user influence diagrams. It contains instructions on the commands and functions included in AFids and how to execute them. For a tutorial on influence diagrams and AFids capabilities with respect to solving problems see the companion document *AFids Tutorial and Example Session.*

AFids was written in Turbo Pascal 4.0 on a Zenith Z-248. It requires MS-DOS version 3.0 or greater, an EGA adapter and monitor, and at least 256K of memory.

A.2 Getting Started

To start the AFids system make sure you are in the subdirectory containing the AFids.mnu file. Two versions of AFids are provided, AFids87.exe is for use with a numeric coprocessor (i.e. an 8087, 80287, or 80387), AFids.exe is for systems without a numeric coprocessor. From the DOS command line type either 'AFids87' or 'AFids' to start the program. If the program will not run make sure that the AFids.mnu file is present and that at least 256K of memory is free. If the program still will not run you may be using an incompatible version of DOS. AFids was compiled under MS-DOS 3.0 and will probably not work with earlier versions.

Once the program starts you will see the greeting screen, to clear it hit any key. Next, you will be presented with the initial preferences screen. You may hit CTRL-ENTER to begin a new diagram or hit 'y' in the first field and enter the file name of an AFids diagram file in the second. Currently, the no-graphics option is not implemented. When the screen reflects your preferences hit CTRL-ENTER to begin.

If you selected a diagram file it will be loaded and the graphics screen will display that portion of the diagram visible from the center of the graphics work area. To begin processing hit any key and the main menu will be presented. You may now perform any AFids command on the diagram.

If you did not select a diagram file for loading you will be taken directly to the main menu where you may construct a new diagram for processing or load an existing one with the Files Menu 'Read' command.

A.3 System Command Structure

Figure 15 shows a typical AFids text screen display. The current menu occupies the bottom three lines of display. The upper lines are used to display diagram information and for data entry screens such as the one in shown.

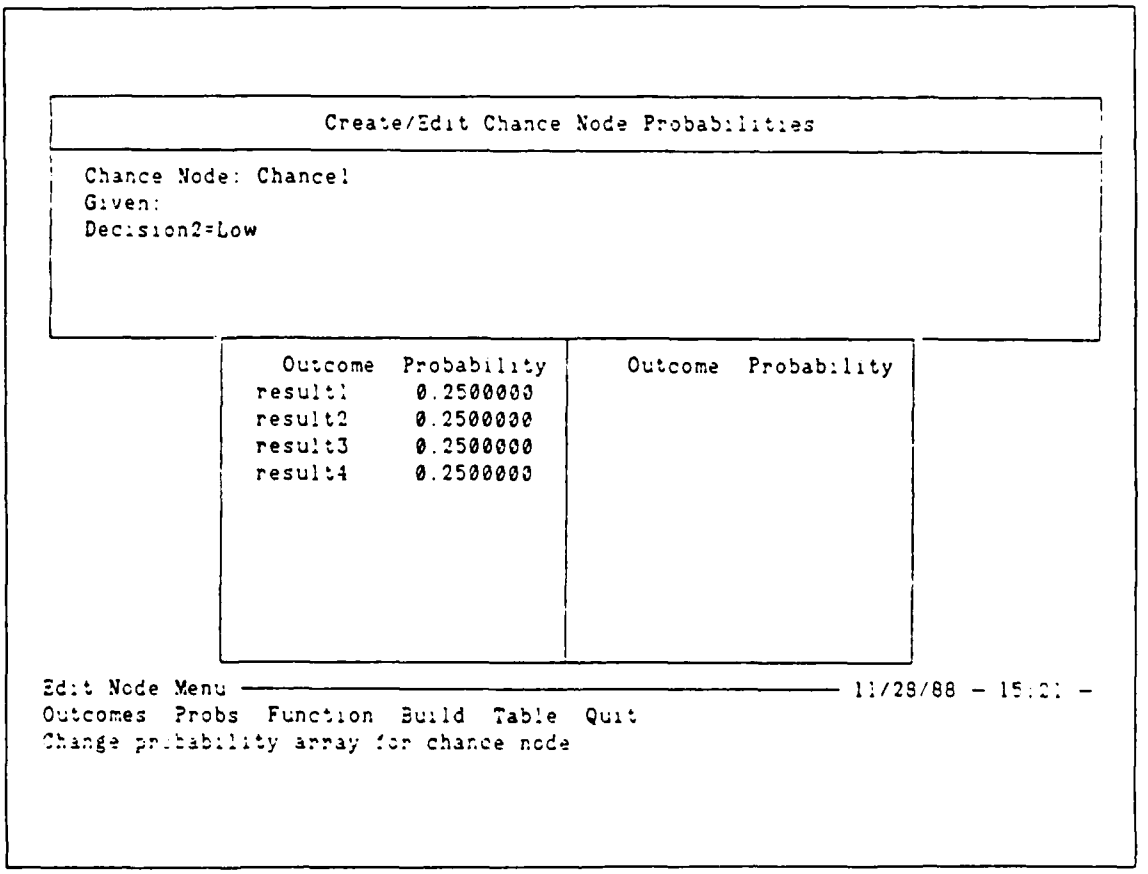

Figure 15. AFids Text Screen Display

The top line of the menu display shows the current menu name and the current date and time. Commands are chosen from second line of the menu by either high-lighting the menu choice and hitting ENTER or by hitting the identifying letter (the first in most cases) of the command name. There will be only one capital letter in a command name this is the identifying letter. Commands may be high-lighted in turn by hitting either the SPACE bar or the left and right arrow keys. The ESC key or the 'q' key will move to a previous menu (or produce a termination prompt if hit at the main menu). Hitting letters other than those identifying commands causes no action. The third line of the menu display is the explanation line. A brief description of the high- lighted command is given in this line.

Above the menu in Figure 15 is a data entry screen. Nearly all user input data in the AFids program will use this type of screen. The data entry screen consists of a number of fields. Data is entered into the field by typing letters and numbers as required. The ENTER key is used to accept the data as it appears in the field. The data currently in the field may be edited using the following command keys:

ENTER Accepts the data as it appears in the field. When data is accepted it is checked for type (i.e. is it a valid integer or real number) and optionally whether it is within a certain range (i.e. probabilities must be ≥ 0.0 and ≤ 1.0). A beep will sound and the cursor will remain in the field if an error is detected. Data in each field must be correct (of the right type) before the cursor can leave the field. The cursor is moved to next field or the whole screen is entered if the cursor is currently in the last field.

CTRL-ENTER Accept the entire screen as it appears. This may be used from any field and can save time hitting ENTER to move down to the last field. This is the recommended way to accept a large number of fields.

TAB, Down Arrow Similar to ENTER, accept the current data and move the cursor to the next field. The cursor will wrap around from the last field to the

first using this command unlike the ENTER key which will enter the whole screen if hit in the last field.

Up Arrow Accept current field and move cursor to the previous field. This command will wrap the cursor from the first field to the last.

Left Arrow, Right Arrow These keys will move the cursor one character within the current field in the indicated direction. A beep will sound if moving past the end of the data is attempted.

END This key will move the cursor to the end or right most character position in the current field.

CTRL-END This key will delete all characters from the cursor position to the end of the field.

HOME This key will move the cursor to the first or left most character position in the current field.

CTRL-HOME This key will delete all characters from the cursor position to the start of the field.

INS Change the character entry mode to Insert. The cursor will change to a large size and any characters entered will be inserted into the string and push characters to the right. Hitting INS again returns to normal Typeover mode.

ESC May be used at any time from any data entry screen to cancel the screen and the operation without harm. No data is ever changed if a data entry screen is exited with the ESC key.

Usually default data is provided for fields requiring entry. Some fields are for display only. Data in these fields cannot be changed. A data entry screen may be exited without changing any data by hitting the ESC key.

Figure 16 shows a typical AFids graphic display. The display shows only 1/9th of the total AFids display work space which is three screens wide by three screens

high. The graphics cursor is a white symbol which looks like $\oplus$. The nodes are represented by the following shapes:

Chance Node Circle or ellipse. Represents a random variable.

Decision Node Square or rectangle. Represents a decision variable.

Deterministic Node Double circle or double ellipse. Represents a function or constant.

Subvalue, Value Nodes Rounded square or rectangle. Represents either a subgoal or the objective function of the model.

Arcs in the diagram are represented by an arrow pointing from the predecessor node to the successor node. In general, the AFids visual representation is the same as most influence diagram literature. To see the entire graphics work space a zoom display is used (See Figure 17). In this mode all objects in the graph are displayed at 1/3 size, nodes are not labeled, nor are arrowheads drawn on the arcs. This mode may be selected from the View menu and is automatically displayed when solving the diagram in visual mode so that the effect of all operations can be observed.

Many commands will require modification to the influence diagram graph. When this is the case the user will be shown the current view of the graph and a graphics cursor will appear. The user positions the graphics cursor with the following keys:

Arrow keys Move the cursor one 'jump' in the indicated direction. If the cursor goes off the screen the screen view will be shifted one half screen in that direction and movement may continue. If the absolute edge of the screen is hit no action will be taken.

Shift-Arrow keys Move the cursor five 'jumps' at a time. This is useful for moving rapidly but may not allow precise positioning.

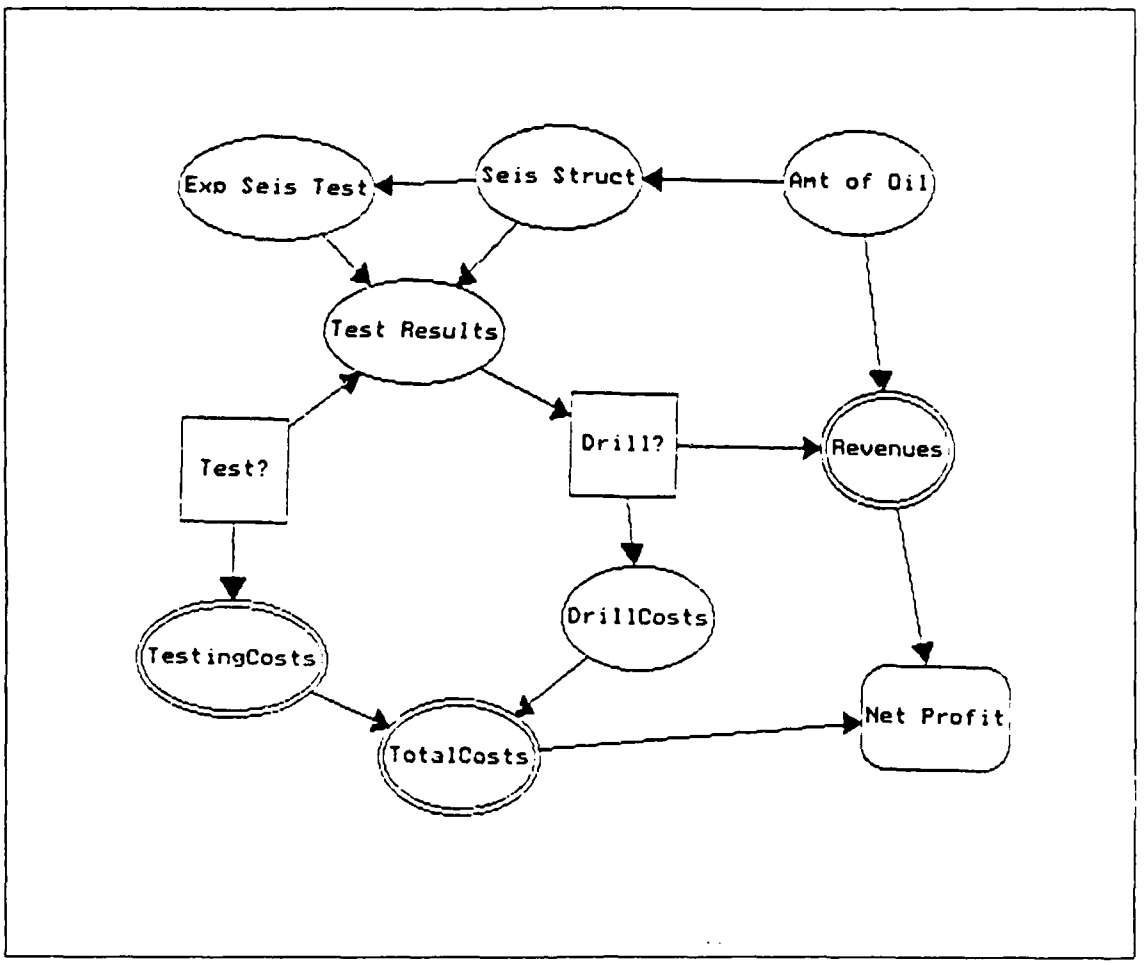

Figure 16. An AFids Graphic Display

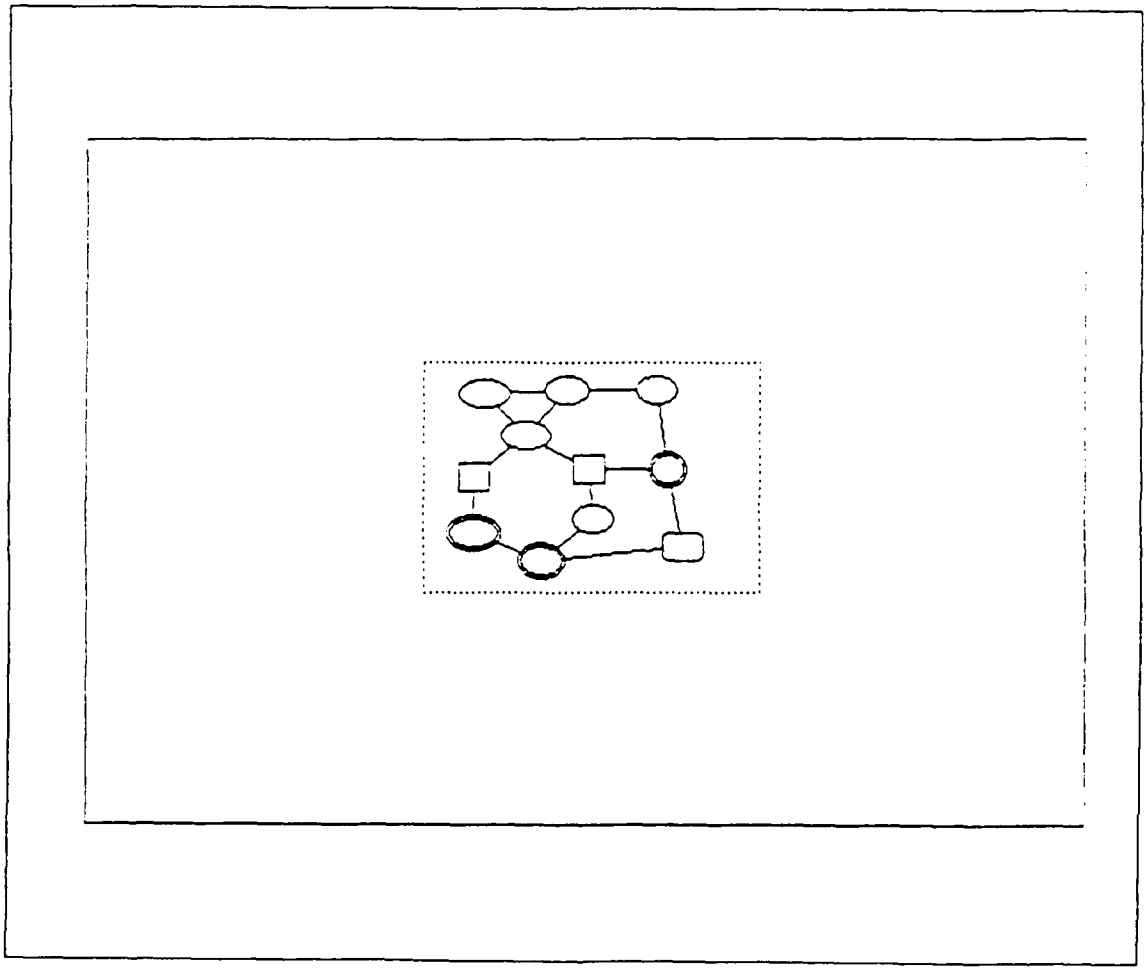

Figure 17. The Zoomed AFids Display

+ Key, - Key Increase or decrease the size of a 'jump'. The default setting is about 10 pixels per jump. This is useful for adjusting the sensitivity of the graphics cursor when using a mouse or when the default jump size is to large or small.

ENTER Return the current position of the graphics cursor. When selecting a node the cursor should be within a distance equal to the size of a normal node(i.e. a circle or square). In some cases when a node has been stretched to hold a very long name hitting ENTER when the cursor is on the extreme edge will not select the node. To select the node simply move the cursor closer to the center of the node and hit ENTER.

ESC This key has two uses depending on the situation. If a single position is to be returned ESC cancels the operation. If multiple nodes are to be selected ENTER selects each one while ESC ends the selection process. Hitting ESC in this case before the first node is selected will cancel the operation.

A.4 Using Dynamic Programming

To use the dynamic programming features of AFids requires no special effort or knowledge on the part of the user. When the command is given to solve the diagram AFids will automatically attempt to separate the value function into subvalue nodes. Once the subvalue nodes are created the diagram will be solved using the dynamic programming algorithm. To separate the value function AFids needs to know that it is separable. Two functions, SUM() and PROD() are known to be separable by AFids. AFids will also split the subvalue nodes if their functions contain a SUM() or PROD() function Splitting the value node requires minimal overhead and the solution using subvalue nodes will always be at least as fast as not using them, so it is recommended that the user utilize the SUM() and PROD() functions whenever an addition or multiplication is in the value function. Subvalue nodes may be created directly during diagram creation if the combined value function will not fit in the

allowed space or if it is simpler to create them explicitly. These subvalue nodes created 'by hand' are treated exactly the same as those created by AFids during solution. Note, if subvalue nodes are the only predecessors of the value node AFids will assume that the value function has been split already and will not attempt to split. This occurs because of AFids ability to be interrupted during the solution process.

A.5 Menu Commands

In the following sections the commands available via the AFids menus are described. As mentioned above, commands may be selected by either high- lighting them and hitting ENTER or by hitting the identifying letter of their name (either upper or lower case letters may be used). The ESC and 'q' keys move to the previous menu. Each menu description will consist of the menu command as it appears in the menu, the identifying letter and a description of its action.

A.5.1 Main Menu

Create/edit [C]

Moves to the Create/Edit menu where diagram creation and editing commands are performed.

Solve [S]

Moves to the Solve menu where diagram solution operations take place.

Files [F]

Moves to the Files menu where file loading, saving, and transcript file control is performed.

Output [O]

Moves to the output menu where the graph may be printed or the data for a node or nodes may be listed to any combination of the screen, a file, or the printer.

New [N]

Causes the current diagram to be erased and the program returns to the initial screen. The user is prompted to verify before the diagram is erased.

sTatus [T]

Displays various information about the current diagram including number of nodes, and available memory.

Hand-calc [H]

Clears the screen and prompts the user for a mathematical expression which is evaluated by the function evaluation procedure. The user enters the expression as a string in algebraic format. Note, only numbers and functions are allowed in the expression, nodes may not be referenced. Expressions are evaluated until a blank line is entered.

View [V]

Moves to the View menu where the graph may be viewed in either full size of zoomed view. Also, the center of the full size view may be adjusted.

Quit [Q]

Prompts the user before quitting the AFids system back to DOS. Note, the current diagram is not saved.

A.5.2 Create/Edit Menu

Add [A]

Move to the Add Node menu where the various types of nodes may be created and placed in the diagram.

Edit [E]

Move to the Edit Node menu where various data items of an existing node may be changed or the outcomes for a deterministic node may be created.

Group [G]

Move to the Group Menu where groups of nodes may be created, edited, moved, and have their outcome limits set.

Delete [D]

Delete a node from the diagram. The graphics display is put up and the user selects the node to remove. The node is destroyed and all of its predecessors with data elements depending on the node are reset. For example all chance predecessors have their probability distributions erased in case they depended on the removed node. These erased data elements must be re- input before solving operations can take place.

Move [M]

Move a node. The graphics display is put up and the user selects a node to move. He then positions the graphics cursor at the new location and hits ENTER. If the node is not too close to another node it is placed there. This selection/placing process continues until ESC is hit.

add-arC [C]

Add an arc to the diagram. The graphics display is put up and the user is prompted to select the predecessor node and then the successor node. An arc is created between them if one does not already exist and if the arc does not create a cycle in the graph.

Rem-arc [R]

Remove an arc from the diagram. The graphics display is put up and the user is prompted to choose the predecessor node and then the successor node. The arc is removed if it exists. Any data elements of the successor node that might have depended on the outcomes of the predecessor node are reset.

Pred-set [P]

Modify the entire set of predecessors for a node. The graphics display is put up and the user is prompted to choose a node. After the node is chosen all

of its predecessors are high-lighted. Predecessors may be added (i.e. an arc created) by selecting a non high-lighted node. A predecessor may be removed by selecting a high-lighted node. When the predecessors are correct ESC is hit to accept the changes or to cancel the command before any changes are made.

Quit [Q]

Return to the Main Menu.

A.5.3 Add Node Menu When commands from this menu are selected the user is presented with a pair of data entry screens to specify the new node's name and either its outcomes or function. When these data have been entered the graphics display is put up and the user is prompted to place the new node in the diagram. To do this, the user moves the graphics cursor to the location for the center of the new node and hits ENTER.

Chance [C]

Add a chance node to the diagram. The user is prompted for the node's name and outcomes. The probability distribution is set from the Edit Node Menu after all of the node's predecessors are determined.

Decision [D]

Add a decision node to the diagram. The user is prompted for the node's name and outcomes. This is all that is required for a decision node.

deTerministic [T]

Add a deterministic node to the diagram. The user is prompted for the node's name and function. For more details on the function definition see Section A.6.

Subvalue [S]

Add a subvalue node to the diagram. This is not normally needed as the solution algorithm will create subvalue nodes as needed but may be useful in special cases or for completely manual operation. The user is prompted for the node's name and function.

Value [V]

Add the value node to the diagram. A value node is needed for solving operations. Only one value node is allowed per diagram. The user is prompted for the node's name and value function. The value function represents the objective function of the model and should be in maximization form i.e. solution operations will attempt to maximize the value of the value node function. If the user wants to minimize a function he should enter the negative of the function so that maximization will drive the value toward zero.

Quit [Q]

Return to the Create/edit Menu.

A.5.4 Edit Node Menu When commands from this menu are chosen the graphics display is put up and the user is prompted to select the node to be modified. To do this the user moves the graphics cursor on the node of interest and hits ENTER. The user is then presented with a data entry screen containing the data currently assigned to the node. The user may then modify the data using the editing commands given above. When the data are correct and the screen is accepted the new data will replace the old. The ESC key may be used to abort the operation at any time.

Outcomes [O]

Modify the outcomes of a node. The node's outcomes are presented in a data entry screen. The user may change the names of the outcomes or their values but may not change the number of outcomes since that could affect successor nodes. To change the number of outcomes for a node delete it and create a new node. This will ensure that all data elements are consistent with the new number of outcomes. The ESC key will exit without change.

Probs [P]

Modify a nodes probability distribution. The probabilities for the node's out-

comes are presented for each combination of predecessor outcomes and the user is allowed to change the probability distribution. The ESC key will exit with no change.

Function [F]

Modify a deterministic, subvalue, or value node function. The function is presented in string form in a data entry screen. The user may modify the function as desired. If the function is modified the outcomes for the node, if calculated, are discarded. For more details on entering and modifying functions see Section A.6.

Build [B]

Build a deterministic, subvalue, or value node's value table from its function. The function is evaluated for each combination of predecessor outcomes and the result stored in the outcome table. For more details on function evaluation see Section A.6. Once this command begins there is no way to halt it short of aborting the program. This is not usually a problem, but evaluating a very large number of predecessor combinations may take several minutes depending on the number of combinations and the complexity of the function.

Table [T]

Build a deterministic, subvalue, or value node's value table by hand. Each combination of predecessor outcomes is presented and the user may enter the value of the node for that combination. This is useful for nodes where no function exists or for which AFids does not possess the capability to evaluate it. This is only useful for nodes with a small number of predecessor outcomes. The ESC key will exit at any without modifying the outcome table if it exists or creating a new one if it does not.

Quit [Q]

Return to the Create/Edit Menu.

A.5.5 Group Operations Menu Commands in this menu are for the manipulation of node groups. Node groups do not have any effect on the data contained in the nodes or the solution process except in the case of outcome-limiting. Groups are displayed on the zoomed graphics display. When the user does selects a command dealing with an existing node group, the user selects the group of interest as follows. When the operation is selected the zoomed graphics display will be put up and the first group of nodes will be high- lighted. The user may cycle through the groups by hitting the SPACE bar. When the correct group is displayed, the user hits ENTER to select it.

Create [C]

Create a new node group. The diagram is displayed in zoom mode. Nodes are high-lighted to enter them in the group. Selecting a high-lighted node de-selects it. When the group is correct the ESC key accepts the group. The operation may be aborted by de-selecting all nodes and hitting ESC.

Edit [E]

Edit a node group. The diagram is displayed in zoom mode and the nodes in the first node group are high-lighted. The SPACE bar cycles through the node groups. Hitting ENTER selects the displayed group for editing. Nodes may then be added or deleted from the group as above by high-lighting and de- high-lighting nodes. The ESC key accepts the modified group. The operation may be aborted by ESC before the group is selected and by hitting ESC without making any changes to the group.

Move [M]

Move a group. The diagram is displayed in zoom mode and the group to move is selected as above. When the group has been selected the graphics cursor is placed at the 'center' of the group i.e. all the locations are averaged. The cursor is then moved to the new center location. The ENTER key will cause the group to be placed around the new center location in the same relative

location as they were around the old center. This is useful for spreading out a diagram which is becoming crowded during creation but is quicker than moving each node individually. The ESC key will abort this operation at any stage.

Set-outcomes [S]

Set the number of outcomes allowed for the node group. As above this command causes the diagram to be displayed in zoom mode and the node groups to be high-lighted with the SPACE bar cycling through them. ENTER selects the group to set outcomes for. The user is prompted to enter an integer number of outcomes that each node in the group may have. This feature only effects deterministic, subvalue, and value nodes. For a detailed discussion on outcome limiting see Section A.6. ESC will abort during group selection. If an incorrect value is entered for the group, the operation may be repeated and the correct value entered.

Quit [Q]

Return to the Create/Edit Menu.

A.5.6 Solve Menu Commands in this menu select the solution mode or otherwise check the diagram for its readiness to be solved.

Manual [M]

Move to Solve Operations Menu. When the Manual command is selected the diagram is checked for missing data. The user is alerted with a error message about each item missing. Solving cannot continue until all required data are present.

Automatic [A]

Solve the diagram automatically. As above this command causes the diagram to be checked for missing data. The user is then prompted to save the diagram if desired. AFids then proceeds to solve the diagram. Upon solution the time elapsed, expected value of the value node, and the optimal policies for the

decision nodes are displayed. This information is also saved to the transcript file and/or printer if active. During automatic solution the keyboard is monitored, if the ESC key is detected the program will abort solving leaving the diagram in a partially solved state. Selecting Automatic or Visual will then restart the solution process where it left off.

Visual [V]

Solve the diagram automatically pausing between solution steps and display the diagram. As above the diagram is checked for missing data. The diagram is displayed in the zoom out mode so that the effect of each solution step may be observed. The diagram is then solved automatically as above. However, the user must hit a key after each step. The ESC key will exit the solution process at any time. This allows the user to inspect nodes, save the partially solved diagram, etc. Solution may then be restarted from where it left off with either Visual or Automatic. This is useful for observing part of the solution process but not allows the user to use the Automatic mode for the remainder. The Visual solution mode is significantly slower than the Automatic mode because of the time required to re-draw the diagram and for the user to hit a key between solution steps. Upon solution the expected value for the model, and the optimal decision policies are displayed.

Show-Missing [S]

Show nodes with missing data. This command performs the same check that the above commands use except that no error messages are generated. Nodes with missing data are high-lighted.

Quit [Q]

Return to the Main Menu.

A.5.7 Solve Operations Menu Using the commands in this menu the user may perform various operations on the influence diagram. All of the operations

performed in the automatic solution algorithm may be duplicated by hand from this menu. Checks are made on each operation before it is done to make sure it is legal. The general format of these operations is:

1. The user selects the command to perform.
2. The graphics display is presented and the user is prompted to select the node or nodes to be operated on.
3. The user selects the nodes (usually the predecessor node first if multiple nodes are required) by positioning the graphics cursor on the node and hitting ENTER.
4. The legality of the operation is checked and if the operation cannot be done an error message describing the problem is given and the user is taken back to step 3.
5. Once the operation checked as okay, it is performed. When the operation is complete the modified graphics display is presented.
6. The program will wait until the user hits any key and then will return to the menu.

The ESC key may be used to abort the operation at any time AFids is waiting for user input. That is, ESC will not abort the calculations for an operation once they are begun.

Reverse [R]

Reverse an arc using Bayes Rule. This command reverses the arc between two chance nodes by using Bayes Rule on their probability distributions. The graphics display is put up and the user is prompted to select first the predecessor and then the successor nodes. The operation is checked to see if it will cause a cycle in the graph. If it is okay to reverse the arc the operation is done

and the modified graph is displayed. The ESC key can be used to abort the operation during node selection.

Expect [E]

Remove a chance node by conditional expectation. This command will remove a chance node by expectation into a subvalue or value node. The graphics display is put up and the user is prompted to select the node to remove. If it is okay to remove the node the operation is done and the modified graph is displayed. The ESC key can abort the operation before the node is selected.

Maximize [M]

Remove a decision node by maximization. This command will remove a decision node by maximizing the value of the subvalue or value node with respect to its outcomes. The graphics display is put up and the user is prompted to select the decision node to remove. The operation is checked, and if it is okay to remove the node the operation is done. The optimal decision policy is stored in the decision node which remains high-lighted in the graph to show that it has been maximized. After the operation the modified graph is displayed. The ESC key may be used to abort the operation before the node is selected.

Combine [C]

Remove a deterministic node by algebraic combination. This operation removes an arc between a deterministic node and a subvalue or value node by substituting the deterministic node's function into that of the subvalue or value node. The node is not actually removed unless it has no other successors besides the subvalue or value successor. The subvalue or value successor inherits the deterministic node's predecessors. The graphics display is put up and the user selects the node to remove. After the operation the modified graph is displayed. The ESC key can abort the operation before the node is selected.

reDuce [D]

Perform algebraic combination on all deterministic predecessors of a node. This

operation will perform the above Combine operation for a certain subvalue or value node until it has no more deterministic node predecessors. This will work if a deterministic predecessor has a deterministic predecessor itself. This operation is useful before general solution is attempted because during the solution process all existing deterministic nodes must be converted to chance nodes. This is a computationally expensive operation while algebraic combination is not. The graphics display is put up and the user is prompted to select the value or subvalue node to reDuce. After the operation the modified graph is displayed. The ESC key can abort the operation before the node is selected.

Split [S]

Split a value or subvalue node. This command will split a subvalue or value node into its components and create new subvalue nodes for each term in its function. The function must be either a *SUM()* or *PROD()*. This command will continue to split the subvalue nodes created until no *SUM()* or *PROD()* functions remain. This operation will automatically place the nodes in the diagram but not in the most esthetically pleasing locations. This function is a key part of the dynamic programming features of AFids. If the function is not separable no action is taken. The graphics display is put up and the user is prompted to select the node to split. After the operation is done the modified graph is displayed. The ESC key can abort the operation before the node is selected.

No forget [N]

Add 'no-forgetting' arcs to the diagram. Arcs from decision nodes and their direct predecessors must be added to all successor decision nodes to correctly solve the diagram. All 'no-forgetting' arcs required are added to the diagram. This may make the diagram very visually messy. This operation may not be aborted. Selecting this command more than once does not cause harm. If no 'no-forgetting' arcs need to be added no action is taken.

Other [O]

Move to the Other Solve Operations Menu. This menu contains several less commonly used solution operations.

View [V]

View the current graph. This command shows the graph display in normal mode. It is useful for choosing the next node to remove.

Quit [Q]

Return to the Solve Method Menu.

A.5.8 Other Solve Operations Menu Commands in this menu follow the same steps as in the menu above except for the View command which simply displays the graph and waits for the user to hit any key.

Integrate [I]

Integrate out one chance node into another. This operation integrates one chance node's probability distribution into that of another. This is not a standard solution operation. The graphics display is put up and the user is prompted to select first the predecessor and then the successor nodes. After the operation the modified graph is displayed. The ESC key may be used to abort the operation during node selection.

reVeal [V]

Reveal the outcome of a chance node. This operation reveals the outcome of a chance node by changing the node into a deterministic node with a constant outcome of the revealed outcome value. The graph is put up and the user is prompted to select the chance node to reveal. The user then enters the outcome which is revealed and the node is transformed into a deterministic node with that value as its constant function. After the operation the modified graph is displayed. The ESC key may be used to abort the operation during the node selection.

Xform [X]

Transform a deterministic node into a chance node. This operation takes a deterministic node and transforms it into a chance node with a probability distribution consisting of probability of 1.0 for the outcome produced by a given combination of predecessor outcomes and 0.0 for all other possible outcomes. This operation is affected by value rounding and outcome limiting as discussed in Section A.6. This operation must be done on all deterministic nodes not removed by algebraic combination before solution of the diagram can begin. The graphics display is put up and the user is prompted to select the deterministic node to transform. After the operation the modified graph is displayed. The ESC key may be use to abort the operation during node selection.

Barren [B]

Remove a barren node from the diagram. If a node is barren (it has no successors) it has no effect on the expected value of the model and may therefore be removed from the diagram. This operation is needed before the diagram is solved and may be needed during solution after a decision node is removed. The graphics display is put up and the user prompted for the node to remove. If the node is barren it is removed. If not, the user is given an error message. The ESC key may be used to abort the operation during node selection.

Quit [Q]

Return to the Solve Operations Menu. This command is not often needed as the program will automatically return to the Solve Operations Menu after an Other Solve Operations command is executed.

A.5.9 Files Menu The commands in this menu are in two groups. First, the storage commands Read and Write deal with influence diagram storage files. These files contain the information necessary to recreate a diagram from scratch. They are ASCII text, but are designed for easy storage and retrieval rather than human reading. These files should not be tampered with since to do so can cause strange

and not always obvious errors in AFids operations. The second group of commands deal with the transcript generated by AFids which records all significant transactions and operations during program execution. Three output 'devices' are provided: a disk file, the text display screen, and the printer (the DOS PRT device). Any combination of these devices may be active and receive transcript information. Note, if a transcript file is not specified at the initial preferences screen it is not available for output. The default settings for the three transcript devices is: transcript disk file ON, screen output ON, printer output OFF.

Read [R]

Read in an AFids diagram file. Nodes read in are added to the current diagram. No checking is currently done for duplicate names or locations. This command is most useful for reading in entire diagrams but may be used for partial diagrams. An error will result if a file containing a value node is read into a diagram already containing a value node. A data entry screen is put up for the user to enter the path and file name of the input file. The ESC key aborts the operation.

Write [W]

Store a diagram to disk. This operation takes the current diagram and stores it to a disk file. Partially solved diagrams may be stored. The output file is in ASCII text but is not designed for human reading. A data entry screen is used to get the path and file name for the storage file. The ESC key aborts the operation.

Transcript [T]

Toggle the transcript file on or off. The transcript file records all operations done on the diagram. If a transcript file was opened at the initial screen selecting this command will prevent data from being written to it. Selecting this command when writing is disabled will enable it again. The user is prompted

to verify what the effect on the transcript file will be. The ESC key may be used to abort.

Screen-echo [S]

Toggle the echo of transcript data to the screen. As above selecting this command will either enable or disable writing of transcript data to the display screen. The user is prompted to verify what the effect will be. The ESC key may be used to abort.

Printer-echo [P]

Toggle the output of transcript data on the printer. As above selecting this command will either enable of disable writing of transcript data to the DOS PRT: device. If no printer is connected the program will hang the first time transcript data is written. The user is prompted to verify what the effect will be. The ESC key may be used to abort.

Quit [Q]

Return to the Main Menu.

A.5.10 Output Menu This menu contains the primary commands for recording the state of the diagram. The Graph command will dump the current graphic display to an Epson compatible printer using any of the six Epson graphic modes. An option for overstrike printing will produce a nicer looking printout but will increase the printing time. The other commands in this menu deal with a text listing of various nodes. All the data AFids has on a given node is displayed in a nice format to the active transcript output devices.

Graph [G]

Print the graphic display. The current graphic display is printed on the Epson compatible printer connected to the PRT: device. The user is prompted for the graphic mode for the printout and whether or not to do double pass printing. Several printouts may be required to print the entire diagram if it does not

fit on one display screen. The ESC key may be used to abort before printing starts.

Diagram [D]

List the entire diagram to the active output devices. The nodes and their data elements are listed to the active combination of screen, transcript file, and printer. This output is formatted for easy reading by the user. This file can not be read in with the Files Menu 'Read' command. When this command is selected the diagram is listed if screen output is active the listing is paused after 20 lines to allow the user to read the output. This command can not be aborted.

Active [A]

List only active (non-removed) nodes. Similar to above but only lists nodes which are currently in the diagram. Removed nodes such as decision nodes removed by maximization are not displayed. This command can not be aborted.

Removed [R]

List only removed nodes. Similar to above but only lists node which have been removed from the diagram. This command can not be aborted.

Nodes [N]

List a user defined set of nodes. Similar to above but a user selected set of nodes is listed. The graphics display is put up and the user is prompted to select a set of nodes to list. The ESC key is used to end the node selection process. The nodes are then listed to the active output devices. Hitting ESC before any nodes are selected will abort the command.

Quit [Q]

Return to the Main Menu.

A.5.11 Graph View Menu Commands in this menu allow the user to view the graph in normal and zoom mode and to chance the center point of the normal

display.

Normal [N]

View the current graph in normal display mode. This mode shows only $1/9^{th}$ of the entire graphics work space. Hit any key to return to the menu.

Zoom [Z]

View the current graph in 'zoom out' display mode. This mode shows the entire graph work space with all diagram objects drawn at $1/3^{rd}$ size. The nodes are shown in their correct shapes but are not labeled, nor are the arrowheads put on the arcs. Hit any key to return to the menu.

Center [C]

Change the center point of the normal graphics mode display. The zoomed display is put up and a white square is drawn to show the location of the current graphic display. The graphics cursor is placed at the center of the square. The user may move the graphics cursor to locate the new center for the normal mode display. Hitting ENTER will cause the square to be redrawn around the current cursor location. The user may adjust the location as often as needed. The ESC key locks in the new normal display center point and returns to the menu.

Quit [Q]

Return to the Main Menu.

A.6 Function Evaluation Subsystem

One of the key features of the AFids package is its ability to handle deterministic nodes and their functions. The AFids package contains the most extensive set of mathematical, logical, financial, and list processing functions of any influence diagram software available. This section will describe the features available for pro-

cessing deterministic functions and for managing the conversion from deterministic to chance node.

A.6.1 Function Format AFids functions are strings of numbers, node names, and function names. A typical function might look like:

SUM(chance1, 5*decision1, det3) * 88.5 * SIN(det4)

Where chance1, decision1, and det3/4 are the names of nodes in the diagram and SUM and SIN are function names. When the function is evaluated the various outcome values for the nodes are substituted into the function wherever their name appears. Functions are entered and edited in a data entry screen. The function data entry screen consists of three fields in which strings may be entered. The fields are not linked so text will not wrap from one to the other if pushed by inserting text. When the screen is accepted the three fields are appended together to form the final function string. A limitation of Turbo Pascal requires that strings be ≤ 255 characters. AFids function strings are limited to about 230 characters to allow room for substituting the node outcome values into the function string during evaluation.

Node names may not be the same as function names. There are no other restrictions on node names. They may contain any character that may be typed from the keyboard including spaces and punctuation symbols. In general short names of $\simeq$5 characters represent a good compromise between a mnemonic description and the space occupied by the node in the graph, as longer names produce bigger nodes. The function evaluation subroutine will be confused by duplicate names and will not evaluate the function correctly. The word 'pi' is also reserved and returns the value of π to the maximum accuracy of the computer.

The word *ALL* may be specified in a function requiring a list of arguments. When the function is evaluated the *ALL* term will be replaced with a list of all of the node's predecessors. This is useful when entering a function before all of the predecessors are known or allowing diagram modification without requiring that the

function be changed. The expression SUM(*ALL*) or PROD(*ALL*) is commonly seen in value functions where dynamic programming will be done.

A.6.2 Value Rounding To reduce the size of the probability array when converting a deterministic node to a chance node, AFids will create only one entry in the outcome table for duplicate outcome values. Normally round off errors will prevent values from being exactly equal. To manage this problem AFids introduces the concept of a round-off value. This value is optionally appended to a function string when it is created. For example: chance1 / decision4 + det2 [3.0]. The term [3.0] in the function is the round off value. The expression inside the square brackets [] can be either a constant or an arithmatic expression containing only constants. References to node names are not When a round-off value is specified for a function its outcomes are restricted to integer multiples of that value. For example, in the function above the outcomes would be rounded to ...,-3,0,3,6,.... This feature allows the user to control the precision of the function calculations. Since only integer multiples of the round-off value are produced by the function the chances that outcome values will be equal, and hence produce a single entry in the outcome table, is increased. This feature is also useful for getting outcomes to convenient precision such as cents with a round-off value of [.01], or to the nearest integer value with [1.0].

While round-off values may reduce the size of the outcome space for a deterministic node, they should be used with care especially when the exact value of a number is important. Special care should be taken when combining several rounded values together as the accumulated round-off error may exceed the magnitude of the answer. Finally, when processing chains of deterministic nodes where the outcomes of one node depend on the outcomes of another deterministic node combinatorial explosion in outcomes can be a problem. Using excessively large round-off values to cure this problem is not recommended as the answer will have little meaning. The recommended method for handling deterministic chains is discussed in the next section.

A.6.3 Outcome Limiting Often in dynamic programming type problems the influence diagram formulation will have a series of deterministic nodes influencing each other in a chain-like manner. Combinatorial explosion in the outcomes of these deterministic nodes can take place because the number of outcomes a given node has is the product of the number of outcomes of all of its predecessors. It is easy to see that a chain of deterministic nodes can quickly require astronomical amounts of storage not to mention processing time. To eliminate this problem AFids introduces the concept of outcome limiting for deterministic, subvalue, and value nodes (the most common application will be to deterministic nodes). Outcome limiting the combinatorial explosion problem of deterministic node chains by only allowing deterministic nodes to have a fixed number of outcomes. Thus, the number of outcomes a deterministic node has is no longer dependent on the number of predecessor outcome combinations but is fixed to some reasonable number before the outcomes are calculated.

In the AFids system when a node is outcome limited the following steps are followed when creating the outcome table:

1. All of the node's possible outcomes are calculated (i.e. one outcome for each combination of predecessor outcomes).
2. The high and low values are used to determine the range of outcome values.
3. The range is divided into the previously set number of outcome values and each raw outcome is rounded to the nearest of the calculated outcome values.

Then when the deterministic node is converted to a chance node, both the outcome table and probability array are reduced in size because only one entry is created for duplicate values.

A.6.4 Available Functions The AFids package has a large number of included functions for use in deterministic, subvalue, and value node functions. This section

will list the functions, their arguments,a brief description, and an example. Specialized functions may be added to AFids by inserting the appropriate subroutine in the function evaluation procedure in the CALC unit and re- compiling the program.

A.6.4.1 Arithmatic Operators

\+ Adds two quantities. $chance + det1 + 1.253$

\- Either negation or subtraction. $-chance1$ or $chance2 - det5$

* Multiplies two quantities. $chance1 * det4$

/ Divides two quantities. Divide by zero is trapped but not in a nice way. $(chance3 + det2)/det6$

^ Raises the first quantity to the second quantity power. Uses $x^y = e^{y \ln x}$ so x must be ≥ 0.0. $det1 \ ^\wedge \ (chance3/decision2)$

A.6.4.2 Logical Operators

& Logical AND of two quantities. Returns 1.0 if both are non-zero and 0.0 if either or both are 0.0. & has the same priority as * and / . $chance3$ & $det2$

| Logical OR of two quantities. Returns 1.0 if either are non-zero and 0.0 if both are 0.0. | has the same priority as + and -. $chance3$ | $det2 * decision1$

=, <> Logical EQUAL (NOT EQUAL) of two quantities. Returns 1.0 if values are exactly equal (not equal) 0.0 otherwise. = and <> are lower priority than + or -. $(chance1 = chance2)$

>, >= Logical GREATER (or GREATER OR EQUAL) of two quantities. Returns 1.0 if the first is larger (larger or equal) than the second. $(det4 >= det3)$

<, <= Logical LESS (or LESS OR EQUAL) of two quantities. Returns 1.0 if the first is less (less or equal) than the second. $det4 < (3*2)$

A.6.4.3 Mathematical Functions In this section the function and its arguments appear in square brackets. These brackets are not entered in the function. The term 'expr' is used to represent a constant or a function of constants and node names.

ABS [ABS(expr)] Absolute Value of a quantity. ABS(*chance*1)

SQRT [SQRT(expr)] Square Root of a quantity. SQRT(*det*2 * *det*3)

SQR [SQR(expr)] Squares a quantity. SQR(99.35+*det*3)

SIN [SIN(expr)] Returns the sine of an angle in radians. SIN($PI/6$)

COS [COS(expr)] Returns the cosine of an angle in radians. COS($-det4$)

TAN [TAN(expr)] Returns the tangent of an angle in radians. SIN(*chance*1) / COS(*chance*1) = TAN(*chance*1)

ATAN [ATAN(expr)] Returns the arctangent of an angle in radians. ATAN(*det*2 + *det*4)

LN [LN(expr)] Returns the natural logarithm of a quantity. The argument must be $\geq$ 0.0. LN(SQRT(*chance*2))

EXP [EXP(expr)] Raises e to the argument power. EXP(*chance*4/*det*3)

FACT [FACT(expr)] Returns the factorial of the argument. The argument is rounded to the nearest integer. Care must be taken not to exceed the maximum value for a real number (1E+38 or 1E+408 for AFids87). FACT(5)

RAND [RAND or RAND(expr)] Returns a random number between 0.0 and 1.0 or between 0.0 and expr. RAND(*chance*3 + 5)

MOD [MOD(expr1, expr2)] Returns the integer remainder of exper1 / expr2. MOD-(5,3)

A.6.4.4 Financial Functions This sections describes the AFids financial functions. No attempt is made to explain the functions beyond their format and arguments. Any of the arguments may be a constant or function.

CV [CV(*principal, rate, periods*)] Computes the compounded value of a principal.

PV [PV(*payment, rate, periods*)] Computes the present value of a series of payments.

FV [FV(*payment, rate, periods*)] Computes the future value of a series of payments.

PMT [PMT(*principal, rate, periods*)] Computes the required payments to repay *principal* in *periods* payments.

RATE [RATE(*future value, present value, periods*)] Computes the necessary rate for *present value* to grow to *future value* in *periods* compounding periods.

CTERM [CTERM(*rate, future value, present value*)] Computes the number of compounding periods required for *present value* to grow into *future value* at *rate* interest rate.

TERM [TERM(*payment, rate, future value*)] Computes the number of payment periods for an investment to reach *future value.*

DDB [DDB(*cost, salvage, life, period*)] Computes the double-declining-balance depreciation allowance for an item for period *period* in *life.*

SLD [SLD(*cost, salvage, live*)] Computes the straight–line depreciation allowance of an item for one period.

SYD [SYD(*cost, salvage, life, period*)] Computes the sum-of-the- years-digits depreciation allowance of an item for period *period.*

A.6.4.5 List Functions The functions listed in this section deal with a list of arguments of any length. The arguments list has the form *arg1, arg2, arg3, ...* The arguments may be a constant or a function. Caution, nesting these list functions more than about three deep can cause a stack overflow condition which will abort the program.

AVG [AVG(*list*)] Computes the average of the values in the list.

CHOOSE [CHOOSE(*control, list*)] Returns the item in the list specified by *control.* The first item in the list is numbered 0.

COUNT [COUNT(*list*)] Returns the number of items in the list.

MAX [MAX(*list*)] Returns the value of the largest item in the list.

MIN [MIN(*list*)] Returns the value of the smallest item in the list.

STD [STD(*list*)] Returns the sample standard deviation for the list.

VAR [VAR(*list*)] Returns the sample variance for the list.

SUM [SUM(*list*)] Returns the sum of the list items. This function identifies a separable function for use in value node separation.

PROD [PROD(*list*)] Returns the product of the list items. This function identifies a separable function for use in value node separation.

A.6.4.6 Miscellaneous Functions This section contains a description of the remaining AFids functions.

IF [IF(*condition, true expr, false expr*)] This function will return the value of *true expr* if the *condition* expression evaluates to a non-zero value. The value of *false expr* will be returned otherwise. Any expression may be used for the *condition* but the logical operators above were the intended way to produce a true/false condition.

UTIL [UTIL(*value, risk tolerance, util=1.0 point*)] Computes the standard decision analysis utility function which is an exponential function with UTIL at 0.0 = 0.0 and UTIL at the *util=1.0 point* = 1.0. This function maps one value measure into another.

VAL [VAL(*util, risk tolerance, util=1.0 point*)] Computes the inverse of the UTIL function. Useful for converting utilities back to the original value measure.

Appendix B. *AFids Program Data Structures*

This appendix contains the Pascal source format of the AFids data structures used to represent the data items making up an influence diagram.

To facilitate logical organization the Pascal 'record' structure was used extensively. This allowed structures composed of several parts to be handled as a single unit. Note, none of the structures given here are allocated at compile time, rather they are allocated from the free memory 'heap' as needed at run time. This allows AFids to have no fixed limits on sizes or numbers of objects.

The primary structure in the program is that for an influence diagram node.

```
id_node = record
            name          : name_string;      { string of length 20 }
            removed       : boolean;
            x_coord       : integer;
            y_coor d      : integer;
            preds         : node_list_ptr;
            succs         : node_list_ptr;
            next          : id_node_ptr;
            node_type     : node_types;
            nouts         : integer;
            outcomes      : outcome_array_ptr;
            opt_alts      : outcome_array_ptr; { for a decision node }
            ngivens       : integer;
            givens        : given_array_ptr;
            nprobs        : integer;
            prob_dist     : prob_array_ptr;
            func_str_ptr  : function_string_ptr
```

```
        end;
```

The structures defining the components of the id_node will now be presented.

```
id_node_ptr = ^id_node;

node_list_ptr = ^node_list;
node_list = record
  node_ptr : id_node_ptr;
  next     : node_list_ptr;
end;

node_types = (chance, decision, deterministic, subvalue, value);

outcome_array_ptr = ^outcome_array;
outcome = record
            name : result_string; { string to hold number }
value : real { either real or IEEE double precision depending
               on the AFids version }
  end;
outcome_array = array [0..3000] of outcome;  { limited by Turbo Pascal }


given_array_ptr = ^given_array;
given = record
          node_name : name_string;
  nouts     : integer;
```

```
  cur_out    : integer;
  outs       : outcome_array_ptr
end;
given_array = array [0..2000] of given;

prob_array = array[0..10900] of real;
prob_array_ptr = ^prob_array;
```

The nodes are made up of these parts created at run time. The arcs in the diagram are represented by entries in the 'preds' and 'succs' linked lists. Note, the pred and succ lists are complementary, that is if a node appears in a pred list it will have the node in whose list it appears in its succ list.

Bibliography

1. Burwell, Capt Thomas M. *PERFORMA A Personal Influence Diagram System for Decision Analysis.* MS thesis, AFIT/GOR/MA/87D 87-2. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.
2. Dreyfus, Stuart E. and Averill M. Law. *The Art and Theory of Dynamic Programming*, Academic Press, New York, 1977.
3. Howard R. A. and J.E. Matheson. *Influence Diagrams, The Principles and Applications of Decision Analysis*, R.A. Howard and J.E. Matheson, Eds. Menlo Park CA: Strategic Decisions Group, 1984, pp. 719-762.
4. Kenley, C.R. *Influence Diagram Models With Continuous Variables.* PhD dissertation. Stanford University, Menlo Park CA, 1986.
5. Owen, Daniel L. "The Use of Influence Diagrams in Structuring Complex Decision Problems", *The Principles and Applications of Decision Analysis*, R.A. Howard and J.E. Matheson, Eds. Menlo Park CA: Strategic Decisions Group, 1984, pp. 765-771.
6. Shachter, Ross D. "Evaluating Influence Diagrams", *Operations Research, 34:* 871-882 (November-December 1986).
7. Shachter, Ross D. and Joseph A. Tatman. "Dynamic Programming and Influence Diagrams", Submitted to *IEEE Transactions on Systems, Man and Cybernetics*, November 1987.
8. Tatman, Joseph A. *Decision Processes in Influence Diagrams: Formulation and Analysis*, Ph.D. dissertation, Stanford University, Menlo Park CA, 1986.
9. Tatman, Joseph A. *Influence Diagrams: A Tutorial*, Class handout distributed in Math 5.70, Decision Analysis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, November 1987.
10. *Turbo Pascal Version 4.0 Reference Manual.* Borland International, Scotts Valley, CA, 1988.

Vita

Captain Christopher T. Baron [redacted] He graduated from high school at Tumwater High School in Tumwater, Washington in June of 1979. He then attended the University of Washington receiving a Bachelor of Science degree in Aeronautics and Astronautics in June of 1983. Upon graduation he was awarded a USAF reserve commission through AFROTC. He was then assigned to the Foreign Technology Division, Wright-Patterson AFB, Ohio and worked in the Soviet Space Employment Group of the Space Systems Division until entering the graduate Operations Research program in the School of Engineering, Air Force Institute of Technology, in June of 1987.

[redacted]

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for Public Release: Distribution Unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GOR/MA/88D-1	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENC	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology (AU) Wright-Patterson AFB, OH 45433	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
INFLUENCE DIAGRAMS: AUTOMATED SOLUTION WITH DYNAMIC PROGRAMMING

12. PERSONAL AUTHOR(S)
Christopher T. Baron, B.S. Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1988 December	98

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	04		Influence Diagrams, Dynamicr Programming, Decision Analysis, Decision Theory

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Advisor: Joseph A. Tatman, Captain, USAF
Assistant Professor, Dept. of Mathematics and
Computer Science

Approved for release in
Accordance with AFR 190-1
10 Jan. 1989

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Joseph A. Tatman, Captain, USAF	22b. TELEPHONE (Include Area Code) (513) 255 - 3098 / 22c. OFFICE SYMBOL AFIT/ENC

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. Abstract
The major goals of this thesis research were to develop a user friendly software package for processing influence diagrams, and to implement in software the extensions necessary for dynamic programming without special action or knowledge on the part of the user. The final goal was to demonstrate the efficiency of the dynamic programming techniques by applying them to several example problems.

A software package, AFids (AFIT influence diagram system) was developed. The system is capable of performance equivalent to the current state of the art in commercial influence diagram software. AFids incorporates the basic influence diagram operations, the separable value function extensions, and an algorithm to automatically solve any properly formed influence diagram. Separation of the value function for dynamic programming is automatic and requires no action or special knowledge on the part of the user beyond representing the value function explicitly as a sum or product. The software uses menus, data entry screens, and graphics to provide an effective and friendly user interface. Several extensions to influence diagram theory were implemented in the AFids package including the concepts of value rounding and outcome limiting to control the combinatorial explosion encountered when processing chains of deterministic nodes. The system is cost free and available in either source or compiled form for government users. There are no restrictions on use or distribution except for commercial use. The software runs on MS-DOS compatible microcomputers and was programmed in Turbo Pascal. A users manual, and a description of the AFids data structures are provided for future users and researchers.

Two application examples are presented, demonstrating both the efficiency of the dynamic programming features and the limitations of influence diagrams in modeling problems with significant functional relations.

AFids provides a solid capability for processing influence diagrams throughout the decision analysis cycle, from formulation to solution. Inclusion of the separable value function and deterministic processing node functions represent advancements in influence diagram software and allow previously computationally intractable problems to be solved via influence diagrams. Finally, AFids will become the standard influence dieagram software for the decision analysis curriculum at AFIT.